USER'S GUIDE

Intelligent Picomotor Control Modules

Driver, Controller, and Joystick

U.S. Patents #5,140,470, #5,168,168, #5,303,035, #5,394,049, #5,410,206

Use of controls or adjustments, or performance of procedures other than those specified herein may result in exposure to high voltages.

NEW FOCUS®

2584 Junction Ave. • San Jose, CA 95134-1902 • USA
phone: (408) 284–6808 • fax: (408) 284–4824
e-mail: contact@newfocus.com • www.newfocus.com

Warranty

New Focus, Inc. guarantees its products to be free of material and workmanship defects for one year from the date of shipment. This warranty is in lieu of all other guarantees expressed or implied and does not cover incidental or consequential loss.

Document Number 872209 Rev. B

Contents

Introduction

Overview

The Picomotor™ is a motorized actuator designed for applications requiring a compact, high-resolution positioner. New Focus offers the Picomotor as a stand-alone actuator as well as integrated into mounts and stages. The Picomotor has a linear resolution of less than 30 nm for Standard Picomotors and less than 100 nm for Tiny Picomotors. The Intelligent Picomotor Driver generates the electronic pulses that provide open-loop control of the Picomotors.

The Intelligent Picomotor control modules—including the Model 8753 Intelligent Picomotor Driver, the Model 8750 Intelligent Picomotor Network Controller, and the Model 8754 Intelligent Picomotor Joystick—offer a compact, rugged, and versatile method for controlling New Focus Picomotors.

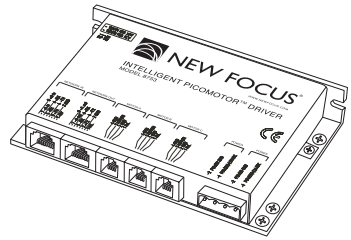
These control modules can be used together for manual control of up to three Model 8753 drivers (nine individual Picomotors), or used with a computer to control up to 31 drivers (93 Picomotors).

The Model 876x Picomotor driver kits include all the components needed for controlling three, six, or nine Picomotors.

Intelligent Picomotor Drivers

Each Model 8753 Intelligent Picomotor Driver can drive up to three individual Picomotors, asynchronously, through its three 4-pin single-channel output ports. (For multi-axis devices that use 6-pin connectors, such as optical mounts, you will need to use a Model 8725 multi-axis adapter.)

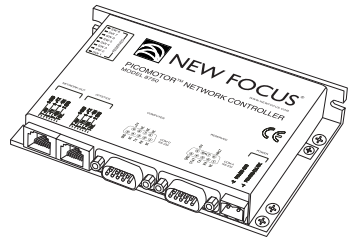
Figure 1:
Model 8753
Intelligent
Picomotor Driver



Each driver is a member of the Distributed Control Network (DCN). Up to 31 DCN devices can be controlled over a multi-drop full-duplex RS-485 network. Standard RJ-45 connectors and commercially available cables are used to connect the drivers into a network. A DLL library is available free of charge on the New Focus web site to facilitate the integration of Picomotor Drivers into your custom applications.

Intelligent Picomotor Network Controllers

Figure 2:
Model 8750
Intelligent
Picomotor Network
Controller



The Model 8750 Intelligent Picomotor Network Controller can be used as an interface between a computer and up to 31 Model 8753 drivers using a standard RS-232 serial interface, or it can be used with the joystick and drivers for manual control of Picomotors. Standard RJ-45 connectors and commercially available cables are used to connect the controller to the drivers and the joystick. The MCL commands described in the "Computer Control: Using MCL" chapter beginning on page 27 can be used to integrate the Picomotor Network Controllers into your custom applications.

Intelligent Picomotor Joysticks

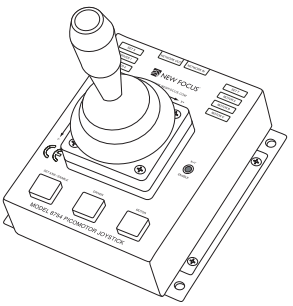


Figure 3: Model 8754 Intelligent Picomotor Joystick

The Model 8754 Intelligent Picomotor Joystick is a multifunction I/O device designed for a wide range of applications. When used with the Model 8750 network controller, it can control up to three Model 8753 drivers (nine Picomotors).

The joystick, like the Intelligent Picomotor driver, is also a member of the DCN, and can be programmed to control any Picomotor on any driver in the DCN using the available DLL library.

Intelligent Picomotor Driver Kits

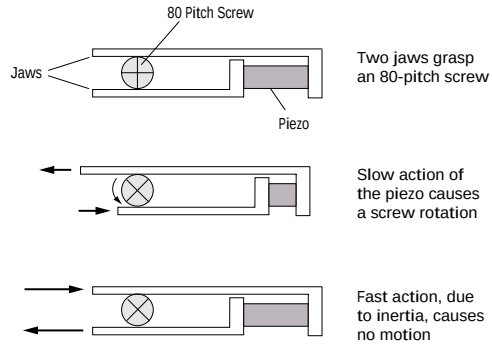
Figure 4: The Model 8766 kit includes a network controller, joystick, and two drivers

The driver kits (shown in Figures 4, 7, and 12) provide an out-of-the-box solution to manually controlling Picomotors. Each kit includes a Model 8750 network controller, a Model 8754 joystick, a power supply, and one, two, or three Model 8753 Intelligent Picomotor Drivers. The kits also include mounting bracket(s) and all required cables. (See “Manual Control: Using the Driver Kits” on page 21.)

General Picomotor Concepts

The key element in our motorized mechanical line is the Picomotor. The Picomotor's revolutionary design relies on the principles of static and dynamic friction. Two jaws grasp an 80-pitch screw, and a piezoelectric transducer slides the jaws in opposite directions.

Figure 5: Schematic of the action of the Picomotor



Slow action of the Picomotor (high static friction) causes the screw to rotate while fast action (low dynamic friction) causes no rotation. By sending pulses with fast rise times and slow fall times, the piezo will rotate the screw counter-clockwise. Similarly, sending pulses with slow rise times and fast fall times rotates the screw clockwise.

Note: *The knob on the end of the drive screw provides inertial mass required for operation. Removal of the knob will prevent the Picomotor from functioning properly and void the warranty.*

The Picomotor drivers generate the 130-V pulses required to drive the piezo in the Picomotor.

Step Size and Repeatability

The Picomotor is not a stepper motor. Because it is a friction mechanism, it does not produce identical steps for each input pulse. Although the step size can vary slightly from pulse to pulse, it will always be less than 30 nm for the Standard Picomotor and less than 100 nm for the Tiny Picomotor.

In addition, there will be a significant variance in step size when starting the motor from a standstill or changing its direction.

Other factors that affect the angle change and linear travel per pulse include direction of rotation, load, temperature, and life and wear of the unit.

Picomotor Products and Accessories

Each driver includes three single-channel 4-pin RJ-22 phone-style output ports. Each Picomotor needs to connect to a separate output port. For devices with 6-pin RJ-11 connectors, you will need to use a Model 8725 multi-axis adapter. The following tables show our standard Picomotor devices and accessories, and lists their connector types or ports where applicable.

Intelligent Picomotor Modules

Model #	Description	Ports
8750	Intelligent Picomotor Network Controller	2 x RJ-45 1 x DB-9M
8753	Intelligent Picomotor Driver	3 x RJ-22 2 x RJ-45
8754	Intelligent Picomotor Joystick	2 x RJ-45

Intelligent Picomotor Kits

Model #	Description
8763	3-Axis Picomotor Driver Kit for versatile motion control: 1 x Model 8750 Intelligent Picomotor Network Controller 1 x Model 8753 Intelligent Picomotor Driver 1 x Model 8754 Intelligent Picomotor Joystick 1 x Model 8760 Driver/Controller Assembly Kit 1 x Model 8755 Driver Kit Power Supply 1 x User's Guide

Model #	Description
8766	6-Axis Picomotor Driver Kit for versatile motion control: 1 x Model 8750 Intelligent Picomotor Network Controller 2 x Model 8753 Intelligent Picomotor Driver 1 x Model 8754 Intelligent Picomotor Joystick 2 x Model 8760 Driver/Controller Assembly Kit 1 x Model 8755 Driver Kit Power Supply 1 x User's Guide
8769	9-Axis Picomotor Driver Kit for versatile motion control: 1 x Model 8750 Intelligent Picomotor Network Controller 3 x Model 8753 Intelligent Picomotor Driver 1 x Model 8754 Intelligent Picomotor Joystick 3 x Model 8760 Driver/Controller Assembly Kit 1 x Model 8755 Driver Kit Power Supply 1 x User's Guide

Intelligent Picomotor Accessories

Model #	Description	Connectors
8721	10' Intelligent Picomotor communication cable (RJ-45)	2 x RJ-45
8722	Intelligent Picomotor communication adapter (RS-232/RS-485)	1 x DB-9F 1 x RJ-45
8723	3' Intelligent Picomotor communication cable (RJ-45)	2 x RJ-45
8724	5" Picomotor communication cable (RJ-45) for daisy chaining drivers or other DCN compatible components	2 x RJ-45
8725	Multi-axis adapter (allows use of multi-axis Picomotor devices with Model 8753 driver)	1 x 6-pin RJ-11 3 x 4-pin RJ-22
8726	6" power cable for daisy chaining DCN modules	2 x 2-pin MSTB-2.5/2-ST-5.08
8727	Power connector for connecting user-supplied power	2-pin MSTB-2.5/2-ST-5.08

Model #	Description	Connectors
8755	Intelligent Picomotor driver kit power supply	2-pin MSTB-2.5/ 2-ST-5.08
8760	Intelligent Picomotor Driver/Controller Assembly Kit (contains Models 8724 and 8726 with hardware for daisy chaining)	2 x RJ-45 2 x 2-pin MSTB-2.5/2-ST-5.08
8761	Intelligent Picomotor Computer Interface Kit (contains Model 8721 cable and Model 8722 adapter)	1 x RJ-45 1 x DB-9F

Picomotors

Model #	Description	Connectors
8301–8341	Standard Picomotor	1 x 4-pin RJ-22
8301v–8341v	Vacuum-Compatible Picomotor	Teflon-insulated 28-gauge wires
8351	Tiny Picomotors	1 x 4-pin RJ-22

Stages and Positioners

Model #	Description	Connectors
8051	Fiber Positioner	1 x 6-pin RJ-11
8071, 8081, 8082	4- and 5-Axis Stages	2 x 6-pin RJ-11
8095	6-Axis Stage	6 x 4-pin RJ-22
8401	Rotary Stage	1 x 4-pin RJ-22

Optical Mounts

Model #	Description	Connectors
8807	Mirror Mount	2 x 4-pin RJ-22
8808–8854	Mirror Mounts	1 x 6-pin RJ-11
888X	Pint-Sized Mounts	2 x 4-pin RJ-22

User Safety

Voltages of up to 130 V are accessible inside the driver chassis, mounts, and Picomotors. Although protection circuits are included, *do not* operate the units with the driver or mount covers removed. If the wire of a mount or Picomotor is frayed, discontinue use and return it for repair.

Setting Up

Overview

The Intelligent Picomotor components—including the Model 8753 driver, the Model 8750 network controller, and the Model 8754 joystick—are a part of a versatile and powerful platform for motion control. Depending upon your unique application, the Intelligent Picomotor network can be set up with the network controller, utilizing it's embedded MCL firmware for manual or computer control, or it can be set up without it, using a PC to directly control the drivers and joystick via DLL functions or the DCN Utility. The different architectures are shown in Figure 6 and Figure 7 below.

Figure 6: Motion control using the Intelligent Picomotor network controller

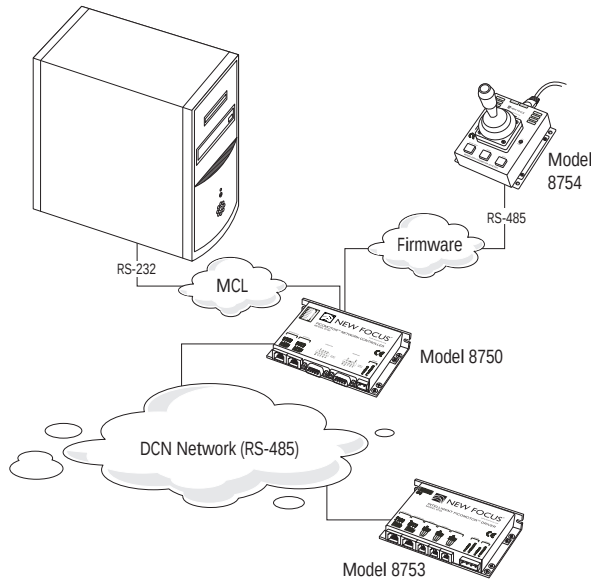
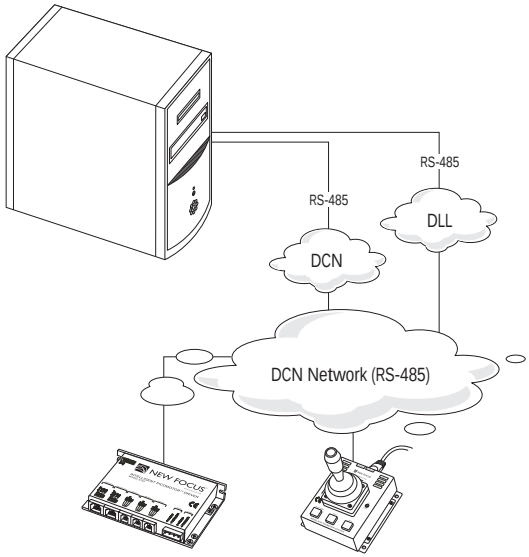


Figure 7: Motion control using the Intelligent Picomotor DCN modules only



Based on your chosen application, the Intelligent Picomotor network can be set up for use in one of four ways:

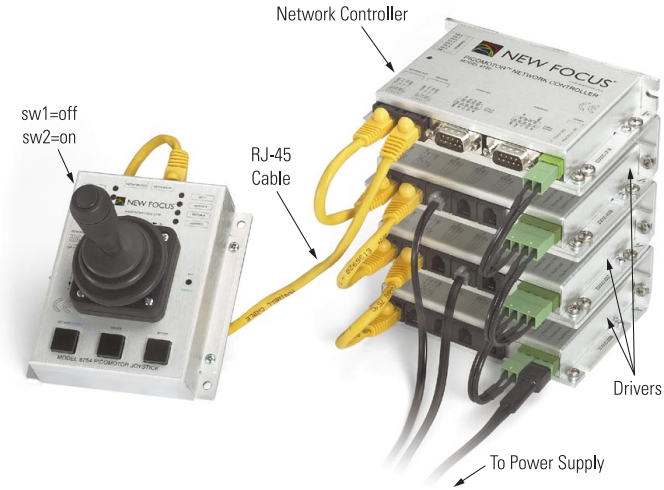
- Manual control using the stand-alone driver kits
- Computer control using MCL with the network controller
- Computer control using DLL/DCN with the driver(s) only
- Computer control using DLL/DCN with the driver(s) and joystick

The following sections describe the four set-up methods and the steps needed to get your system up and running.

Stand-Alone Driver Kits

The Model 876x Intelligent Picomotor Driver Kit is an out-of-the-box solution for manual motion control. The kits come pre-assembled, with the network controller and driver(s) mounted together, and include all of the cables and adapters needed for connecting the controller, driver(s), joystick, and power supply. The standard hardware configuration for the Model 8769 Driver Kit is shown in Figure 8.

Figure 8: Standard Hardware Configuration for Model 8769 Stand-Alone Kit



The following section takes you through the basic steps for connecting the components of the kit. The controls and functions are described in the "Manual Control: Using the Driver Kits" chapter beginning on page 21.

Note: *If you purchased the Intelligent Picomotor modules individually, rather than in a kit, then they will need to be assembled first using the Model 8760 Driver/ Controller Assembly Kit.*

1. Connect the Model 8750 network controller's **Network Out** port to the Model 8753 driver's **Network In** using a 5" communication cable (Model 8724).
2. If you are using more than one driver, connect the **Network Out** from the first driver to the **Network In** on the next driver using another 5" communication cable.
3. Repeat Step 2 for all remaining drivers.
4. For the last driver on the network, set the two terminating resistors' DIP switches to the "ON" position.
5. Connect the network controller's **Joystick** port to the joystick's **Network In** using a Model 8723 communication cable.

Note: *The joystick can actually be hooked up to either network port on the controller, not just the **Joystick** port.*

6. Verify that the joystick's DIP switches are set to their default positions: **SW1** is "OFF" and **SW2** is "ON."
7. Connect the controller's power supply connector to the first driver's power connector using the Model 8726 power supply cable. Be sure to place the power supply cable connector into the right two pins in the driver.
8. If you are using more than one driver, use the additional Model 8726 power cables to connect power. Align the cables such that the connected pins from the adjacent drivers match, i.e., if the cable comes from the right pins it plugs in to the right pins of the next driver, then the next cable goes out from the left pins and plugs into the left pins of the next connector, and so on (see Figure 8).
9. Connect the power supply to the remaining power input on the last driver in the chain (i.e., the driver furthest away from the controller).

Note: *Each power supply supports up to three drivers. If you have more than three drivers in your installation, you will need to hook in additional power.*

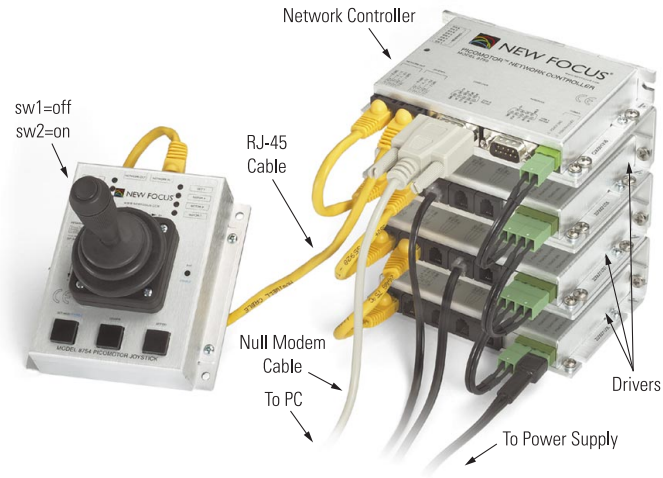
Using MCL with the Network Controller

The MCL firmware for the New Focus network controller contains a simple but powerful set of host commands that can be used to integrate the network controllers into your custom applications. The MCL commands are described in the "Computer Control: Using MCL" chapter beginning on page 27.

To access the MCL commands, set up your network as described in the "Stand-Alone Driver Kits" section above. Then, using a standard DB-9 NULL-Modem RS-232 cable, connect the **COM** port of your host computer to the **Computer** connector on the network controller. The baud rate should be set to 19200; there are eight data bits, one stop bit, and no parity.

Figure 9 shows the hardware configuration for using MCL.

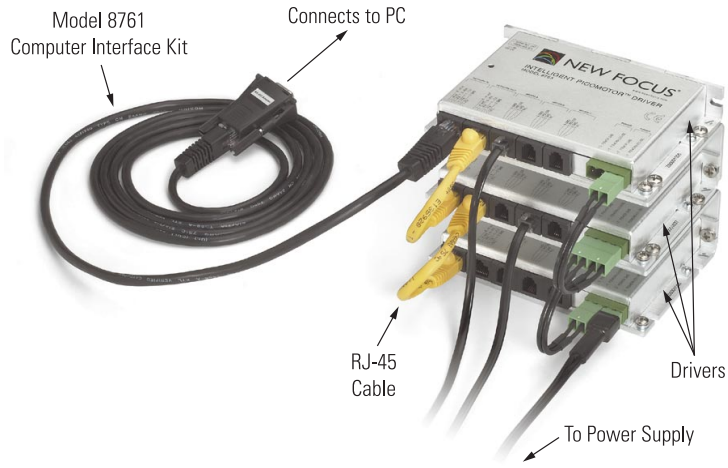
Figure 9: Hardware Configuration for Using MCL



Using DLL/DCN with the Driver(s) Only

For computer control of the drivers using the available DLL library (page 63) or DCN Utility (page 53), you will need to build your system without using the network controller. The hardware configuration is shown in Figure 10.

Figure 10: Hardware Configuration for Using DLL/DCN with Driver(s) Only



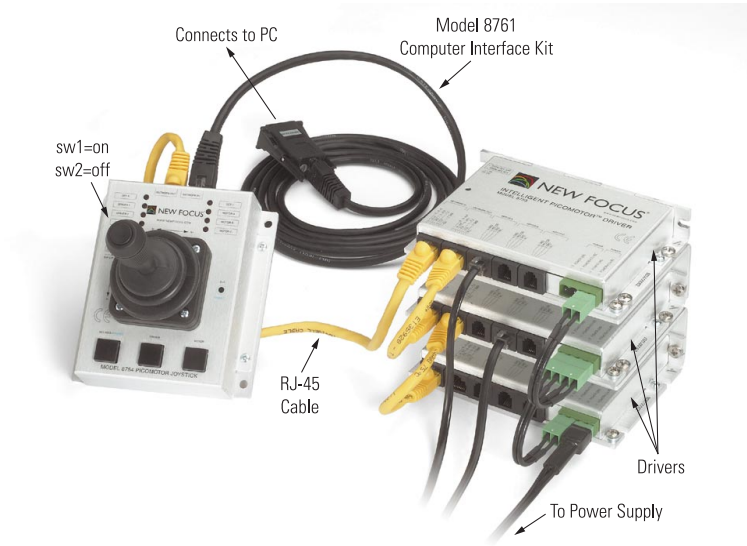
1. Use the Model 8761 Computer Interface Kit to connect the **Network In** port of the driver to the computer's **COM** port.
2. If you are using more than one driver, connect the **Network Out** port of the first driver to the **Network In** on the next driver using the Model 8724 communication cable.
3. Repeat Step 2 for all remaining drivers.
4. For the last driver on the network, set the two termination resistors' DIP switches to the "ON" position.
5. Connect the Model 8755 power supply to the power connector on the driver.
6. If you are using more than one driver, connect the power supply to the last driver on the network and daisy chain adjacent drivers with the Model 8726 power cable. Align the cables such that the connected pins from the adjacent drivers match, i.e., if the cable comes from the right pins it plugs in to the right pins of the next driver, then the next cable goes out from the left pins and plugs into the left pins of the next connector, and so on (see Figure 10).

Note: *Each power supply supports up to three drivers. If you have more than three drivers in your installation, you will need to hook in additional power.*

Using DLL/DCN with the Driver(s) and Joystick

The DLL library (page 63) and DCN Utility (page 53) also include commands for the joystick. To utilize them, you will need to build your system without using a network controller. The hardware configuration is shown in Figure 11.

Figure 11: Hardware Configuration for Using DLL/DCN with Driver(s) and Joystick



1. Set the DIP switches on the joystick as follows: **SW1** is "ON" and **SW2** is "OFF."
2. Use the Model 8761 computer interface kit to connect the **Network In** port of the joystick to the computer's **COM** port.
3. Connect the **Network Out** port of the joystick to the **Network In** port of the driver using a Model 8723 communication cable.
4. If you are using more than one driver, connect the **Network Out** port of the first driver to the **Network In** on the next driver using the Model 8724 communication cable.
5. Repeat Step 2 for all remaining drivers.
6. For the last driver on the network, set the two termination resistors' DIP switches to the "ON" position.
7. Connect the Model 8755 power supply to the power connector on the driver.
8. If you are using more than one driver, connect the power supply to the last driver on the network and daisy chain adjacent drivers with the Model 8726 power cable. Align the cables such that the connected pins from the adjacent drivers match, i.e., if the cable comes from the right pins it plugs in to the right pins of the next

driver, then the next cable goes out from the left pins and plugs into the left pins of the next connector, and so on (see Figure 11).

Note: *Each power supply supports up to three drivers. If you have more than three drivers in your installation, you will need to hook in additional power.*

Manual Control: Using the Driver Kits

Overview

This chapter describes using any of the Model 876x driver kits. It also applies to any setup that includes a joystick and a network controller.

Figure 12: The Model 8763 kit includes a network controller, joystick, and one driver

What's Included

Each kit includes a Model 8750 Intelligent Picomotor Network Controller, a Model 8754 Intelligent Picomotor Joystick, a Module 8755 power supply, and one, two, or three Model 8753 Intelligent Picomotor Drivers. The kits also include mounting bracket(s) and all required cables.

Control Modes

The network controller contains firmware that allows you to control the Picomotor drivers in three modes:

- **Stand-alone mode:** The joystick controls the motors on up to three drivers.
- **Edit mode:** The joystick controls the motors on up to three motors, with additional parameters edited using the MCL computer interface (see page 27).
- **Command mode:** The joystick is disabled, and the MCL computer interface controls the motors on up to 31 drivers (see page 27). When the network is switched to command mode, all of the LEDs on the joystick will be illuminated.

Rules of Operation

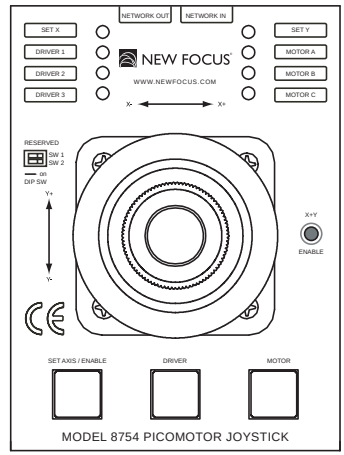
When setting up and using the Intelligent Picomotor Driver Kits, it is important to note the following:

- The joystick must always be present at power-up for the network controller to function. An LED will flash on the network controller to indicate when the joystick is missing or not connected properly.
- Once the network is initialized, the joystick can be unplugged and replugged into the network. Note that when the joystick is replugged into the network, it does a reset, loads saved values, and goes into stand-alone mode automatically.
- The network controller has a battery backed-up RAM which is normally enabled (see "Network Controller DIP Switches" on page 121.) This allows the joystick settings (I/O states) to stay in memory when the system is powered off.
- Drivers can also be unplugged and replugged into the network, but the driver and channel settings are set to default whenever this is done. The network should be reset after adding or removing a driver.
- If additional drivers are added so that there are more than three drivers on the network, the joystick will still be able to access only the first three drivers in the stand-alone mode. The additional drivers can be accessed/controlled by using MCL commands under computer control (see "Computer Control: Using MCL" on page 27).

- Changes made to parameters in edit mode stay current until a power cycle or initialization reloads the "saved" values. To make any change permanent, parameters must be saved (page 34).
- The default control mode after power-up or reset is stand-alone.
- Pressing the **Set/Axis Enable**, **Driver**, and **Motor** buttons on the joystick will reset the network and load the saved parameters. This will work even when the joystick is turned off by a `J0F` MCL command.
- Pressing the **Set/Axis Enable**, **Motor**, and **X+Y/Enable** buttons on the joystick will reset the network and load the default parameters. This will work even when the joystick is turned off by a `J0F` MCL command.

Using the Joystick

Figure 13: Top view of Model 8754 joystick



Enabling/Disabling the Joystick

You can disable the joystick to prevent accidentally changing the position of Picomotors or of joystick settings. To enable or disable the joystick:

1. Press and hold the **Set Axis/Enable** button. While this button is held down, moving the joystick will not move any Picomotors.
2. Press and release the **X+Y/Enable** button on the top of the joystick.
3. Release the **Set Axis/Enable** button.

While the joystick is disabled, all three driver LEDs will be illuminated on the joystick.

Selecting the Motors to Control

The Model 8754 is a two-axis joystick that can control up to nine Picomotors. By default, the X axis and Y axis will be set to the first and second motor ports on Driver 1. Use the buttons on the joystick to change the assignment of motors to the X axis and Y axis of the joystick.

1. Press the **Set Axis/Enable** button to select which axis you want to change. The Set X and Set Y LEDs at the top of the joystick will indicate which axis is open for programming.

   You can prevent accidental changes to joystick settings by pressing **Set Axis/Enable** until both of these LEDs are off. In this state, the LEDs will tell you the combination of the drivers/motors selected for the X and Y axes.

2. Press the **Driver** button to select a driver for the axis. The Driver 1, Driver 2, and Driver 3 LEDs will indicate the active driver.

   The driver numbers represent the order in which the drivers are connected to the Network Controller, with Driver 1 being the one directly connected to the controller.

3. Press the **Motor** button to assign a different motor of the selected driver to the axis. The Motor A, Motor B, and Motor C LEDs will indicate the selected axis. You will not be able to assign the same driver and motor to both the X and Y axes of the joystick.

   To disable an axis, press the **Motor** button until all three motor LEDs are off.

By default, the network controller firmware is set to control Standard Picomotors. If you have selected any Tiny Picomotors to control, you must change the motor type settings using MCL commands (see "Set Motor Type" on page 46) before using the joystick. A Picomotor can be damaged if it is driven with the wrong type of driver signal for an extended period of time, so it is important to ensure that the motor driver is configured to generate the correct driver signals.

Controlling Picomotors

Once you have assigned motors to the joystick, moving the joystick handle will run the selected motors.

- For the X axis, moving the joystick left of center will move the Picomotor counter-clockwise or backwards; right of center results in clockwise or forward motion.
- For the Y axis, moving the joystick above center will move the Picomotor clockwise or forward; below center results in counter-clockwise or backward motion.
- The proportional behavior of the joystick is dependent on the velocity, minimum velocity, and acceleration control parameters. See "Setting Velocity and Acceleration Parameters" on the next page for more information.

The joystick is fully proportional, so the farther the joystick is moved from the center, the faster the motor will move. The joystick can also be configured to run in a "bang-bang" mode by setting the velocity and minimum velocity appropriately.

Note: *To prevent accidental motion, there is a "dead zone": moving the joystick less than 5° from center will not cause any motion.*

Setting Velocity and Acceleration Parameters

Using the Motion Control Language (MCL), you can set parameters that will allow better control over your motorized devices. Although the drivers support a maximum speed of 2 kHz, you can use MCL to set minimum velocity, maximum velocity, and acceleration—see "Computer Control: Using MCL" on page 27. The joystick will automatically use these settings.

For example, to reconfigure the joystick for "fine" speed control (e.g., 1 Hz to 10 Hz), the velocity of the particular channel needs to be changed and saved (to make the change permanent). See the MCL "Example 3" on page 52.

You can use the joystick to reset the velocity, acceleration, and speed factor parameters:

- Press the **Set Axis/Enable**, **Driver**, and **Motor** buttons simultaneously to revert to the last saved parameters.
- Press the **Set Axis/Enable**, **Driver**, and **X+Y/Enable** buttons simultaneously to revert to the factory default parameters (maximum velocity = 2 kHz, minimum velocity = 8 Hz, and acceleration = 8 steps/s^2).

Moving Two Motors Simultaneously

Although each driver can only drive one Picomotor at a time, the joystick does support driving two Picomotors simultaneously, as long as they are attached to different drivers.

1. Assign the X axis and Y axis of the joystick to Picomotors on different drivers. (See "Selecting the Motors to Control" on page 24.)
2. Press and hold the **X+Y/Enable** button on top of the joystick to move the two motors simultaneously.
3. Release the **X+Y/Enable** button to move just one axis.

Computer Control: Using MCL

Overview

The MCL firmware for the New Focus network controller contains a simple but powerful set of host commands. It can control up to 31 Picomotor drivers and a joystick. The communication protocol is strictly master-slave protocol—the host sends a command to the Network Controller and receives the answer. An answer can be either success or failure.

The Network Controller maintains two sets of operational parameters—default parameters and non-volatile (Flash) parameters. On power up, the firmware will load the Flash parameters. MCL lets you override the existing firmware or save new values for various parameters.

Using the RS-232 Interface

The host computer can communicate serially with the network controller using a standard DB-9 NULL-Modem RS-232 cable connected to the **Computer** connector on the network controller. The baud rate should be set to 19200; there are eight data bits, one stop bit, and no parity.

The first driver should be connected to the **Network Out** connector. The joystick should be connected to the **Joystick** connector.

Note: *See "Using MCL with the Network Controller" on page 16 for hardware set-up instructions.*

Software Set-Up

MCL commands can be sent to the network controller either by using a standard Windows Hyperterminal and manually typing in individual commands or by using custom programs like LabVIEW™, C/C++, or Visual Basic® to send sequences of commands.

Hyperterminal Settings

To set up Windows Hyperterminal for communication with the Network Controller, you will need to take the following steps:

1. Open a Hyperterminal session.
2. Under the **File** menu, select **Properties**.
3. In the **Connect To** tab, select the proper serial port from the "Connect using" list box.
4. Click the **Configure** button and set the following parameters:

Bits per second	19200
Data bits	8
Parity	NONE
Stop Bits	1
Flow control	NONE

5. Close the Port settings window by clicking **OK**.
6. Select the **Settings** tab.
7. Verify the following settings:

Emulation	AUTODETECT
Backscroll buffer lines	500

8. Click the **ASCII Setup** button.
9. Set the following parameters:

   Echo typed characters locally
   Append line feeds to incoming line ends

   All other check boxes should not be selected.
10. Close the ASCII Setup window by clicking **OK**.
11. Close the Properties window by clicking **OK**.

Note: *You may experience problems with line feeds in some versions of Hyperterminal. Alternate software is available; see the Network Controller Terminal section below.*

Network Controller Terminal

A stand-alone program for communicating with the MCL firmware, called Network Controller Terminal, can be downloaded from the New Focus web site. Network Controller Terminal functions like a Windows Hyperterminal and can be used to send individual MCL commands via a serial port to the Picomotor network controller.

To use Network Controller Terminal:

1. Hook up the network controller to the **COM** port of your PC using a NULL-modem cable.
2. Start the Network Controller Terminal program.
3. Verify the "COM port" setting matches the hardware connection. "COM1" is the default setting.
4. Press the **Initialize** button. If initialization is successful, the "Command" text field will change from gray (inactive) to white (active).

   If initialization is not successful, check that your hardware connections, cable type, and COM port settings are correct. See "Using MCL with the Network Controller" on page 16 for more information.
5. To use the software, type commands into the "Command" text field and press **Send**. Responses will appear in the "Terminal window."

Rules of Operation

In addition to the rules of operation defined in the previous chapter (page 22), you need to keep the following in mind when using MCL:

- MCL commands can be sent even when the joystick is unplugged or is not present at power-up.
- INI is equivalent to a pressing the **Set/Axis Enable**, **Driver**, and **Motor** on the joystick.
- INI plus DEF is the same as the **Set/Axis Enable**, **Motor**, and **X+Y/ Enable** buttons on the joystick.
- The saved values are loaded upon power-up or reset. A `DEF` command must be issued to return to the default values.

- The parameter settings for the stand-alone and command modes are self-contained. For example, when you switch from command to stand-alone mode, the joystick returns to the state it was last in when in stand-alone mode.
- You must always turn the joystick off (issue a JOF command) if you want to move the motors in command mode. If this is not done, MCL will try to start motion and the joystick will stop the motion, and the motors will only move a few pulses. Note that this is true even if the joystick is unplugged.
- Whenever an MCL command (e.g., MPV, VEL, or ACC) is issued for a motor channel different from the currently selected one, the motor channel is automatically changed to the new one.

Programming for the Network Controller

When programming for the network controller, keep the following rules in mind:

- All drivers on the network have unique addresses. A driver's address contains the letter 'A' and a number designating its position in the network. For example, the driver connected to the network controller will be A1. (The device address is not case sensitive.)
- Each driver can support up to three motors, numbered 0, 1, and 2, where 0=Motor A, 1=Motor B, and 2=Motor C.

  Note: *When responding to a query, the network controller designates the motors as M0, M1, and M2.*

- The joystick has four digital inputs (buttons numbered 0 to 3) and eight digital outputs (LEDs numbered 0 to 7). The analog inputs are expressed in (x,y) coordinates of the joystick axis.

Network Controller Responses

The communication protocol is master-slave. The host sends a command to the network controller and the network controller sends back an answer. There are two types of answers:

- **Acknowledgment:** Upon successful completion of a command, there is a <Carriage return><Line feed><Greater than sign>.

- **Non-acknowledgment:** If there is a problem with the command syntax, the network controller will return <Carriage return><Line feed><Question mark>.

  For example, the following commands will cause errors:

  ```
  >ain 0
  VALUE IS OUT OF RANGE - PARAMETER 1
  ?
  ```

  ```
  >mp
  UNKNOWN COMMAND
  ?
  ```

Conventions

The following pages contain a summary of all available commands, followed by detailed definitions for each command. The following conventions are used in both the "Command Summary" and the "Command Definitions" sections.

- The commands are case insensitive.
- Values to be input are indicated by angle brackets (<>) and are separated from the command by a space.
- Optional values are indicated by square brackets ([]).

Command Summary

Common Commands

Syntax	Command	Page
DEF	Load Default Parameters	33
INI	Initialize Devices on Network	33
SAV	Save Parameters	34
VER	Query Firmware Version	34

Picomotor Control Commands

Syntax	Command	Page
ACC [<driver>] [<motor>]	Query Motor Acceleration	35
ACC <driver> <motor>=<value>	Set Motor Acceleration	36
CHL [<driver>]	Query Motor Channel	37
CHL <driver>=<motor>	Set Motor Channel	37
FOR <driver> [=<value>]	Set Direction to Forward	38
GO [<driver>]	Start Motion	38
HAL [<driver>]	Stop Motion Smoothly	39
MOF [<driver>]	Disable Motor Driver	39
MON [<driver>]	Enable Motor Driver	40
MPV [<driver>] [<motor>]	Query Minimum Profile Velocity	41
MPV <driver> <motor>=<value>	Set Minimum Profile Velocity	42
POS [<driver>]	Query Motor Position	43
REL <driver>=<value>	Set Relative Position	43
REV <driver>[=<value>]	Set Direction to Reverse	44
STA [<driver>]	Query Device Status	44
STO [<driver>]	Stop Motion	45
TYP [<driver>] [<motor>]	Query Motor Type	45
TYP <driver> <motor>=<type>	Set Motor Type	46
VEL [<driver>] [<motor>]	Query Motor Velocity	46
VEL <driver> <motor>=<value>	Set Motor Velocity	47

Joystick Control Commands

Syntax	Command	Page
AIN [<axis>]	Query State of Analog Inputs	47
IN [<button>]	Query State of Digital Inputs	48
JOF	Turn Off Stand-Alone Mode	49
JON	Turn On Stand-Alone Mode	49
OUT [<LED>]	Query State of Digital Outputs	50
OUT <LED>=<state>	Set State of Digital Output	50

Command Definitions

Common Commands

Load Default Parameters

Syntax `DEF`

Description Loads the default parameters:
velocity = 2 kHz, acceleration = 32,000 steps/s^2,
minimum profile velocity = 8 Hz.

Example
```
>def
>
```
(The default parameters are loaded.)

Initialize Devices on Network

Syntax `INI`

Description Initializes all the devices on the network. This command is also executed after power-up.

Example
```
>ini
>
```
(All of the devices on the network are initialized.)

Save Parameters

Syntax	SAV
Description	Saves velocity, acceleration, minimum profile velocity and motor type parameters for all drivers.
Example	>sav > (The operational parameters are saved.)

Query Firmware Version

Syntax	VER
Description	Returns the firmware version.
Example	>ver Version 1.0.13 > (The controller is running firmware version 1.0.13.)

Picomotor Control Commands

Query Motor Acceleration

Syntax	`ACC [<driver>] [<motor>]`
Description	Returns the acceleration for all motors, for the three motors of a specified driver, or for a specified motor on a particular driver.
Argument	Driver: A1 to A31 Motor: 0 to 2
Response	"<driver> <motor>=x" x = 16 to 32000 Units: steps/s^2

Example 1	`>acc`
	`A1 M0=20000`
	`A1 M1=20000`
	`A1 M2=20000`
	`A2 M0=10000`
	`A2 M1=10000`
	`A2 M2=10000`
	`>`
	(There are two drivers on the system, A1 and A2. All motors on A1 are set to accelerate at 20,000 steps/s^2; all motors on A2 are set to accelerate at 10,000 steps/s^2.)

Example 2	`>acc a1`
	`A1 M0=20000`
	`A1 M1=20000`
	`A1 M2=20000`
	`>`
	(The motors on driver A1 are all set to accelerate at 20,000 steps/s^2.)

Example 3	`>acc a2 2`
	`A2 M2=10000`
	`>`
	(The acceleration for motor 2 on driver A2 is 10,000 steps/s^2.)

Set Motor Acceleration

Syntax	`ACC <driver> <motor>=<value>`
Description	Sets the acceleration of a specified motor. The channel is changed automatically when issued while in command mode. *Note: If a* ***VEL****,* ***MPV****, or* ***ACC*** *command is issued while in command mode, then these become the current values for the joystick and stay current until saved or default values are loaded by* ***INI****,* ***DEF****, reset or power-up.*
Argument	Driver: A1 to A31 Motor: 0, 1, or 2 Value: 16–32,000 Units: steps/s^2

Example	Set the acceleration for motor 1 on driver A2 to 10,000 steps/s^2:
	`>acc a2 1=10000`
	`>`

Query Motor Channel

Syntax	`CHL [<driver>]`
Description	Returns the selected motor channels for all the drivers or for a single specified driver. *Note: Each driver can support up to three motors, but only one motor channel can be selected at a time.*
Argument	Driver: A1 to A31
Response	"<driver>=x" x = 0, 1, or 2

Example 1	`>chl`
	`A1=0`
	`A2=1`
	`>`
	(Motor channel 0 is selected on driver A1, and motor channel 1 is selected on driver A2.)

Example 2	`>chl a2`
	`A2=1`
	`>`
	(Motor channel 1 is selected on driver A2.)

Set Motor Channel

Syntax	`CHL <driver>=<motor>`
Description	Sets the motor channel for a specified driver. *Note: Each driver can support up to three motors, but only one motor channel can be selected at a time.*
Argument	Driver: A1 to A31 Motor: 0, 1, or 2

Example	Set the motor channel to 1 for driver A1:
	`>chl a1=1`
	`>`

Set Direction to Forward

Syntax	`FOR <driver>[=<value>]`
Description	Presets the specified driver so it will move forward with either the currently set velocity or a specified velocity.
Argument	Driver: A1 to A31 Value: 1 to 2000; should be greater than MPV Units: Hz

Example 1	Set A1 to move forward with the current speed:
	`>for a1`
	`>`

Example 2	Set A1 so it will move forward with a speed of 1500 Hz:
	`>for a1=1500`
	`>`

Start Motion

Syntax	`GO [<driver>]`
Description	Starts the currently selected motor (for all drivers or only a specified driver) using the previously defined trajectory parameter. *Note: Before issuing this command, you should turn the joystick off (`JOF`). You should also select a channel (`CHL`), turn on the motor driver (`MON`), set the velocity (`VEL`), and set a motor command (`REL`, `FOR`, or `REV`).*
Argument	Driver: A1 to A31

Example 1	Start the motion of all motors:
	`>go`
	`>`

Example 2	Start the motion of the selected motor on driver A2:
	`>go a2`
	`>`

Stop Motion Smoothly

Syntax	`HAL [<driver>]`
Description	Smoothly stop all motors, or just the motor on the specified driver, with the preset acceleration.
Argument	Driver: A1 to A31

Example 1	Stop the movement of all motors:
	`>hal`
	`>`

Example 2	Stop the movement of the motor on driver A1:
	`>hal a1`
	`>`

Disable Motor Driver

Syntax	`MOF [<driver>]`
Description	Turns off all motor channels on all drivers or on the selected driver. You can still set parameters for disabled drivers, but the `GO` command will be ignored.
Argument	Driver: A1 to A31

Example 1	Turn off all drivers:
	`>mof`
	`>`

Example 2	Turn off driver A1:
	`>mof a1`
	`>`

Enable Motor Driver

Syntax	`MON [<driver>]`
Description	Enables all drivers or the specified driver. A driver must be enabled before you can run any motors attached to that driver.
Argument	Driver: A1 to A31

Example 1	Enable all connected drivers:
	`>mon`
	`>`

Example 2	Enable driver A1:
	`>mon a1`
	`>`

Query Minimum Profile Velocity

Syntax	`MPV [<driver>[<motor>]]`
Description	Returns the Minimum Profile Velocity (MPV) parameter for all the drivers, a specified driver, or a specified motor. *Note: In the stand-alone mode, a higher MPV results in a larger dead zone of the joystick. An MPV near the velocity will create a "bang-bang" mode for the joystick, where the joystick will only move the motor at full velocity, and only when the joystick is set near the limit of its travel. In the command mode, a higher MPV will reduce the acceleration time to achieve a specified velocity.*
Argument	Driver: A1 to A31 Motor: 0, 1, or 2
Response	"<driver> <motor>=x" x = 0 to 1999 Units: Hz

Example 1	`>mpv`
	`A1 M0=8`
	`A1 M1=8`
	`A1 M2=8`
	`A2 M0=8`
	`A2 M1=8`
	`A2 M2=8`
	`>`
	(The MPV for all motors on all drivers is 8 Hz.)

Example 2	`>mpv a1`
	`A1 M0=8`
	`A1 M1=8`
	`A1 M2=8`
	`>`
	(The MPV for all motors on driver A1 is 8 Hz.)

Example 3	`>mpv a1 0`
	`A1 M0=8`
	`>`
	(The MPV for motor 0 on driver A1 is 8 Hz.)

Set Minimum Profile Velocity

Syntax	`MPV <driver> <motor>=<value>`
Description	Sets the MPV for the specified motor. The channel is changed automatically when issued while in command mode. *Note: In the stand-alone mode, increasing the MPV will increase the dead zone of the joystick. Setting the MPV near the velocity will create a "bang-bang" mode for the joystick, where the joystick will only move the motor at full velocity, and only when the joystick is set to the limit of its travel. In the command mode, a higher MPV will reduce the acceleration time to achieve a specified velocity.* *Note: If the velocity is set to a value less than the MPV, the controller will automatically set MPV to MPV=velocity-1.* *Note: If a `VEL`, `MPV`, or `ACC` command is issued while in command mode, then these become the current values for the joystick and stay current until saved or default values are loaded by `INI`, `DEF`, reset or power-up.*
Argument	Driver: A1 to A31 Motor: 0, 1, or 2 Value: 0 to 1999 (integer), must be less than velocity. Units: Hz

Example	Set the MPV for motor 0 on driver A1 to 1 Hz:
	`>mpv a1 0=1`
	`>`

Query Motor Position

Syntax	`POS [<driver>]`
Description	Returns the number of pulses sent to the motor since the last motion command. You can query the pulses for active motors on all drivers or on a specified driver.
Argument	Driver: A1 to A31
Response	"<driver>=x" x = -2147483648 (0x80000000) to +2147483647 (0x7FFFFFF)

Example	`>pos`
	`A1=1990`
	`A2=0`
	`>`
	(There have been 1990 pulses sent to the active motor on driver A1 since the last command; 0 pulses have been sent to the active motor on driver A2.)

Set Relative Position

Syntax	`REL <driver>=<value>`
Description	Sets the number of steps to move forward or back to move the active motor on the specified driver.
Argument	Driver: A1 to A31 Value: -2147483648 (0x80000000) to +2147483647 (0x7FFFFFF)
Example	Set the active motor on driver A1 to move 2500 steps clockwise: >rel a1=2500 >

Set Direction to Reverse

Syntax	`REV <driver>[=<value>]`
Description	Presets the driver to move in reverse with either the current speed or a specified speed.
Argument	Driver: A1 to A31 Value: 1 to 2000; should be greater than MPV Units: Hz

Example	Preset driver A1 to move in reverse with a speed of 1500 Hz:
	`>rev a1 = 1500`
	`>`

Query Device Status

Syntax	`STA [<driver>]`
Description	Returns the status bytes of all drivers or just that of a specified driver.
Argument	Driver: A1 to A31
Response	*See the "Driver: Status Byte" on page 107 for status byte descriptions.*

Example	`>sta`
	`SYSTEM STATUS: 0x0`
	`A1=0x3D`
	`A2=0x1C`
	`NO ERROR, READY`
	`>`
	(The status byte of each driver is returned.)

Stop Motion

Syntax	`STO [<driver>]`
Description	Abruptly stops the motion of active motors on all drivers or just the motor on a specified driver.
Argument	Driver: A1 to A31

Example	Stop the motion of the active motor on driver A1:
	`>sto A1`
	`>`

Query Motor Type

Syntax	`TYP [<driver>] [<motor>]`
Description	Returns the motor type setting for the selected channel on all the drivers, for the selected channel on a specified driver, or for the specified channel on a driver. *Note: This query returns only the motor type setting for a channel. The actual type of motor that is connected may be different.*
Argument	Driver: A1 to A31 Motor: 0, 1, or 2
Response	"0" for Standard Picomotor; "1" for Tiny Picomotor

Example	`>typ a1 0`
	`0`
	`>`
	(Motor channel 0 on driver A1 is set to Standard motor type.)

Set Motor Type

Syntax	`TYP <driver> <motor>=<type>`
Description	Sets the motor type for a specified channel. *Note: This only sets the motor type in the controller settings. The actual type of motor that is connected may be different.*
Argument	Driver: A1 to A31 Motor: 0, 1, or 2 Type: 0 = Standard Picomotor, 1 = Tiny Picomotor

Example	Set motor 0 on driver A1 to Tiny motor type:
	`>typ a1 0=1`
	`>`

Query Motor Velocity

Syntax	`VEL [<driver>] [<motor>]`
Description	Returns the velocity in velocity mode or goal velocity in trapezoidal mode. You can query the velocity for all motors, for the three motors of a specified driver, or for a specified motor on a particular driver. *Note: If a `VEL`, `MPV`, or `ACC` command is issued while in command mode, then these become the current values for the joystick and stay current until saved or default values are loaded by `INI`, `DEF`, reset or power-up.*
Argument	Driver: A1 to A31 Motor: 0, 1, or 2
Response	"<driver> <motor>=x" x = 0 to 2000 Units: Hz

Example	`>vel a1 2`
	`A1 M2=2000`
	`>`
	(The velocity of motor 2 on driver A1 is 2000 Hz.)

Set Motor Velocity

Syntax	`VEL <driver> <motor>=<value>`
Description	Sets the velocity for the specified motor. The channel is changed automatically when issued while in command mode.
Argument	Driver: A1 to A31 Motor: 0, 1, or 2 Value: 1 to 2000; should be greater than MPV Units: Hz

Example	Set the velocity of motor 1 on driver A1 to1000 Hz:
	`>vel a1 1=1000`
	`>`

Joystick Control Commands

Query State of Analog Inputs

Syntax	`AIN [<axis>]`
Description	Returns the values of both analog inputs (X and Y coordinates) or just that of a specified analog input corresponding to the current joystick position.
Argument	Axis: 1 or 2
Response	“I1=x” or “I2=y” x = the x coordinate of the joystick position y = the y coordinate of the joystick position Range: 0 to 255

Example

```
>ain
I1=131
I2=127
>
```

(The x and y coordinates of the joystick position are (131, 127).)

```
>ain 2
I2=127
>
```

(The y coordinate of the current joystick position is 127.)

Query State of Digital Inputs

Syntax

```
IN [<button>]
```

Description

Returns the states of all the digital inputs (buttons) or that of a specified button on the joystick.

Argument

Button:

0 = **Set Axis/Enable** button
1 = **Driver** button
2 = **Motor** button
3 = **X+Y/Enable** button

Response

"<button>=y"

Button = 0 to 3

y = "0" for off, "1" for on

Example

```
>in
I0=0
I1=1
I2=0
I3=0
>
```

(Button 1 is on; buttons 0, 2, and 3 are off.)

Example

```
>in 2
I2=0
>
```

(Button 2 is off.)

Turn Off Stand-Alone Mode

Syntax	`JOF`
Description	Turns off the joystick stand-alone mode, and all LEDs on the joystick turn on. The joystick is put into command mode, where the controller will respond to MCL commands from a PC. In this mode, the joystick inputs (buttons) will not function as in the stand-alone mode but can act as user inputs for a user-defined program.

Example	Turn off stand-alone mode:
	`>jof`
	`>`

Turn On Stand-Alone Mode

Syntax	`JON`
Description	The joystick is set to stand-alone mode, and the controller will respond to the user pressing various buttons on the joystick as defined in "Using the Joystick" on page 23. The default mode on power-up is the stand-alone mode.

Example	Turn on stand-alone mode:
	`>jon`
	`>`

Query State of Digital Outputs

Syntax	`OUT [<LED>]`
Description	Returns the states of the all the digital outputs (LEDs) or just that of a specified LED on the joystick.
Argument	LED: 0 = Set X LED 1 = Driver 1 LED 2 = Driver 2 LED 3 = Driver 3 LED 4 = Set Y LED 5 = Motor 1 LED 6 = Motor 2 LED 7 = Motor 3 LED
Response	"<LED>=y" LED= O0 to O7 y = "0" for off, "1" for on

Example	`>out 2`
	`O2=0`
	`>`
	(The LED 2 is off.)

Set State of Digital Output

Syntax	`OUT <LED>=<state>`
Description	Sets the state of the specified digital output (LED) on the joystick. *Note: The output state cannot be changed in stand-alone mode.*
Argument	LED: 0 to 7 State: 1 turns on the LED; 0 turns off the LED
Example	Turn on LED 4: >out 4=1 >

Examples

Example 1

To first disable the joystick and then drive a Standard Picomotor hooked up to driver A2, motor B, in velocity mode at 500 Hz clockwise (forward) with minimum velocity of 0 and acceleration of 5000 steps/sec^2, and then enable the joystick back on, the sequence of commands will be as follows:

```
>jof
>chl a2=1
>typ a2 1=0
>mpv a2 1=0
>vel a2 1=500
>acc a2 1=5000
>mon
>pos
A1=0
A2=0
>for a2
>go
>sto
>pos
A1=0
A2=1832
>jon
```

Example 2

To simultaneously drive two Tiny Picomotors hooked up to driver A1, motor A, and driver A2, motor B, for 5000 steps counterclockwise (backward) and 10000 steps clockwise (forward), respectively, in position mode at a velocity of 2000Hz and with default minimum velocity and acceleration values, the sequence of commands will be as follows:

```
>jof
>def
>chl a1=0
>chl a2=1
>typ a1 0=1
>typ a2 1=1
>vel a1 0=2000
>vel a2 1=2000
>mon
>pos
A1=0
A2=0
>rel a1=-5000
>rel a2=10000
>go
>pos
A1=-5000
A2=10000
>jon
```

Example 3

To permanently set the maximum velocity in the stand-alone mode to be 10 Hz and to achieve proportional control from 1 Hz to 10 Hz for a particular channel on a particular driver (e.g., motor A on driver A1), the sequence of commands will be as follows:

```
>mpv A1 0=0
>vel A1 0=10
>sav
```

Computer Control: Using the DCN Interface

Overview

The New Focus Picomotor DCN Set-Up and Diagnostic Utility allows users to test the basic functionality of the Picomotor driver and joystick modules. The utility is written in Visual Basic and uses DLL functions to communicate with the Picomotor driver and joystick modules.

The DCN Set-Up and Diagnostic Utility can be found on the New Focus web site.

Note: *This program cannot be used with the Model 8750 network controller.*

Using the RS-485 Interface

Driver(s) Only

If your installation uses Picomotor drivers only, the first driver in the network needs to be connected to one of the **COM** ports of a PC using the New Focus Model 8761 Intelligent Picomotor Computer Interface Kit *(see "Intelligent Picomotor Accessories" on page 10).*

Note: *See "Using DLL/DCN with the Driver(s) Only" on page 17 for more detailed set-up instructions.*

Driver(s) and Joystick

If your installation includes a joystick along with the Picomotor drivers, the joystick needs to be connected to the COM port of a PC using the Model 8761 interface kit. The Picomotor drivers should then be connected to the joystick.

Note: *The dipswitch settings of the joystick need to be changed from their default positions.*

Note: *See "Using DLL/DCN with the Driver(s) and Joystick" on page 18 for more detailed set-up instructions.*

Rules of Operation

When setting up the DCN Utility for communication with the driver(s) and joystick, you will need to keep the following in mind:

- When starting the DCN Utility for the first time (or if the "Dcn.ini" file is missing), the first found COM port is chosen by default. A list of connected modules should appear on the left.
  - If the wrong COM port is chosen by default, select the correct one. The network will automatically reset.
  - If a COM port is not in the list of available ports, another application may have control of it, or it may not exist. Close the other application or check your hardware configuration as required and restart the DCN Utility.
  - If no modules are found, re-check your connections, make sure logic power is supplied to all the modules, and verify that all modules have had the proper terminator settings.
  - If some but not all modules are found, re-check your connections and reset the network manually using the Reset Network button. The Reset Device button can be used to reset the currently selected module instead of the entire network.
- The default baud rate for communication is 19200; there is no real need to operate at other baud rates except to test the hardware reliability at higher communication rates.

- With a number of DCN modules connected to one of the **COM** ports, this utility will search for modules and initialize them with addresses starting at 1 for the first module. The list on the left side of the DCN Utility window will show all of the modules found on the network, along with their assigned addresses, types, and version numbers.
- Clicking on one of the modules in the module list will cause that module's properties to be displayed in the control panel on the right. If a Picomotor driver is selected then the Picomotor Driver Control Panel will be displayed, and if a Joystick module is selected the Joystick Control Panel will be displayed.
- When the DCN Utility is started for the first time, each of the modules will be programmed with default operating values during initialization. These default values will also be displayed in the various fields on the Control Panel.
- To exit the program, click on the Windows **X** or press the **Exit** button.

Note:

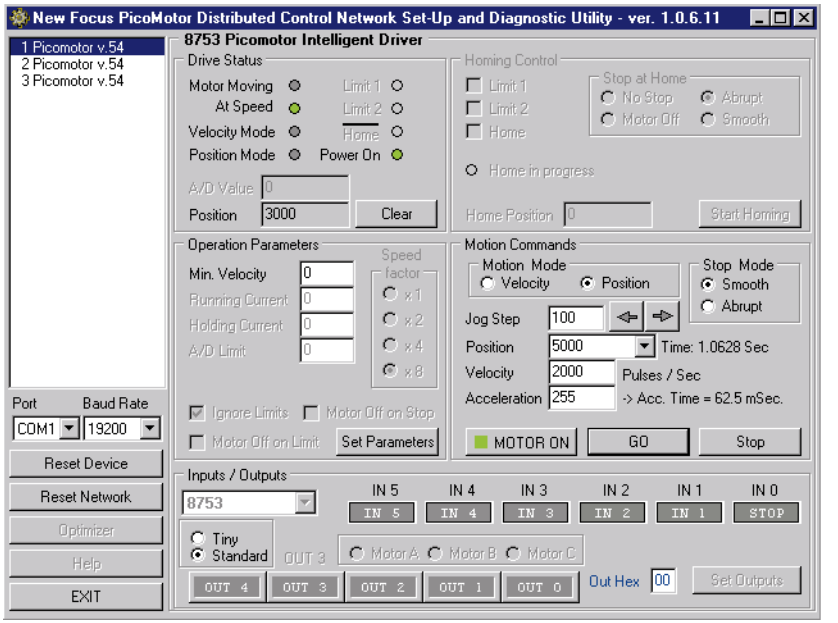
*On exit or re-initialization (when the **Reset Network** button is pressed) the operating parameters for all the modules will be saved in the "DCN.ini" file and the next initialization of the network will load those parameters. The user can also define custom sets of parameters in separate ".ini" files, which can be loaded by right clicking on the **Reset Network** button.*

Picomotor Driver Control Panel

The Picomotor Driver Control Panel, shown in Figure 14, is displayed when a Picomotor driver is selected from the module list displayed on the left side of the DCN Utility window.

The various options available on the Picomotor Driver Control Panel are described in the following sections. A typical sequence of steps that needs to be executed to achieve motion is described on page 60.

Figure 14: Picomotor Driver Control Panel



Drive Status Panel

The Drive Status panel gives information about the operating status as reported directly by the Picomotor driver. A green LED will be set in the following cases:

- Motor Moving: The motor is moving and is cleared otherwise.
- Power On: The motor power output signal is on.
- At Speed: The commanded velocity is reached.
- Velocity Mode: The motor is moving in velocity mode.
- Position Mode: The motor is moving in position (trapezoidal) mode.

Position

The Position represents the state of the 32-bit internal position counter, which indicates the number of pulses output since the last time the counter was reset. The position counter has a range of ±2,147,483,647 steps.

The counter is reset to zero during power-up, when the program is first loaded, or if the **Clear** or **Reset Device** buttons are pressed. The Position can be reset to zero only when the motor is not in motion.

Operation Parameters Panel

The Operation Parameters panel allows the user to specify the minimum velocity. This value should always be less than or equal to the Velocity value specified in the Motion Commands panel (see below). The units of the Min. Velocity are Hz (pulses/second), and the range of allowed values is 0 to2000 Hz.

Motion Commands Panel

The Motion Commands panel allows the user to specify motion parameters and to start and stop a Picomotor. The various parameters are defined below.

Motion Mode

In "Position" mode, the motor moves with a calculated trapezoidal velocity trajectory from its starting position to the target position. The range of position in motor steps (or pulses) is ±2,147,483,647. The specified Acceleration will be used, and the specified Velocity will not be exceeded. The target Position can either be an absolute signed (+ve or –ve) position entered in the Position field and initiated by the **GO** button, or a relative position entered in the Jog Step field, with the → (forward or CW) or ← (backward or CCW) buttons used to define direction and initiate motion.

In "Velocity" mode, the velocity profiler is used to accelerate or decelerate the motor from its Min. Velocity to the specified target Velocity, which can be positive or negative. To change the Velocity value, a stop command must first be issued by pressing the **Stop** button. Velocity values less than the Min. Velocity, as defined in the Operation Parameters panel, will not be accepted.

Stop Mode

If the Stop Mode is set to "Smooth," then upon pressing the **Stop** button, the motor will decelerate to a stop at the specified acceleration

rate. If the Stop Mode is set to "Abrupt," the motor will stop immediately at its current position when the **Stop** button is pressed. The Stop mode is only applicable in the Velocity Motion Mode.

Jog Step

Available only in the Position Motion Mode, the Jog Step acts as a relative-move command. The number of pulses can be entered in the text field, and the left and right arrows next to it can be used to initiate the motion in CW (forward) and CCW (backward) direction, respectively. The arrows do not need to be held down once the motion has started. The units are pulses and the range is ±2,147,483,647 pulses.

Position

The Position field is also available only in the Position mode. It indicates the absolute position to move to with the **GO** button. The units are pulses with a range of ±2,147,483,647 pulses.

Velocity

The Velocity field is applicable to both the Position and Velocity Modes and indicates the peak velocity in both cases. The units are Hz (pulses/second), with a range of 1 to 2000 Hz. The Velocity cannot be smaller than the Min. Velocity value.

Note: *The Velocity value is internally transformed to an 8-bit velocity number in conjunction with a speed factor. As a result, in some cases, the resultant velocity will be close to but not exactly equal to the user-defined velocity value. For example, velocity values of 2000 or 1000 Hz will result in exactly the specified pulse output frequencies, but velocity values of 1900 or 1100 will result in slightly different pulse output frequencies.*

Acceleration

This is an acceleration factor, with a range of 1 to 255, which is used to determine both acceleration and deceleration in both the Position and the Velocity modes. The actual acceleration time can be calculated based upon the equations defined in "Velocity Profile Mode" on page 105.

MOTOR ON

The **MOTOR ON** button turns the driver amplifier on or off. If the amplifier is on, first a stop command is sent to the Picomotor drive to stop the motor in the specified manner and then the amplifier is disabled. If the amplifier is off, this button turns the amplifier on.

GO Button

The **GO** button is used to start the motion after the relevant parameters have been defined. This button is used in Velocity mode and absolute Position mode (using the Position text field, not the Jog Step field).

Stop Button

The **Stop** button will send the Picomotor drive a stop command for the selected stopping mode. The amplifier is kept enabled. This button is typically used only in Velocity Mode; in Position Mode, the motion will automatically stop once the specified number of pulses has been sent to the Picomotor. However, if the **Stop** button is pressed in the middle of a move in Position Mode, the motion will stop immediately without having reached the target position.

Other Information

Other information, such as the time for motion in Position Mode with absolute move and the acceleration time, are also displayed. The time is reset to 0 after the target position has been reached.

Inputs/Outputs Panel

The Inputs/Outputs panel lets the user select the motor channel for the currently selected driver, along with the type of Picomotor—Tiny or Standard—connected to that channel.

The motor type or channel can be changed only when the driver is off. If the driver is on, press **MOTOR ON** to switch it off.

The Inputs/Outputs panel also displays some of the input byte and IO byte bits for your reference.

Typical Operation Sequence

A typical sequence of steps that needs to be executed to achieve motion, after the network has been successfully initialized, is listed below:

1. Select a particular Picomotor driver from the module list displayed on the left side of the DCN Utility window.
2. Choose the motor channel (e.g., Motor A) and motor type (e.g., Standard) in the Inputs/Outputs panel.
3. Set Min. Velocity (e.g., 1) in the Operations Parameters panel.
4. In the Motion Commands panel:
   - Choose the Motion Mode (e.g., Velocity).
   - Select the Stop Mode (e.g., Smooth).
   - Type a Velocity value (e.g., 1000 pulses/sec).
   - Type an Acceleration factor value (e.g., 255).
5. Press **Clear** (in the Drive Status panel) to reset the position to 0.
6. Press **MOTOR ON** (in the Motion Command panel).
7. Press **GO**. The motor will start moving and the pulse count will be shown in the Position field in the Drive Status panel.
8. Press **Stop** to stop motion. The total number of pulses output is indicated in the Position field in the Drive Status panel.

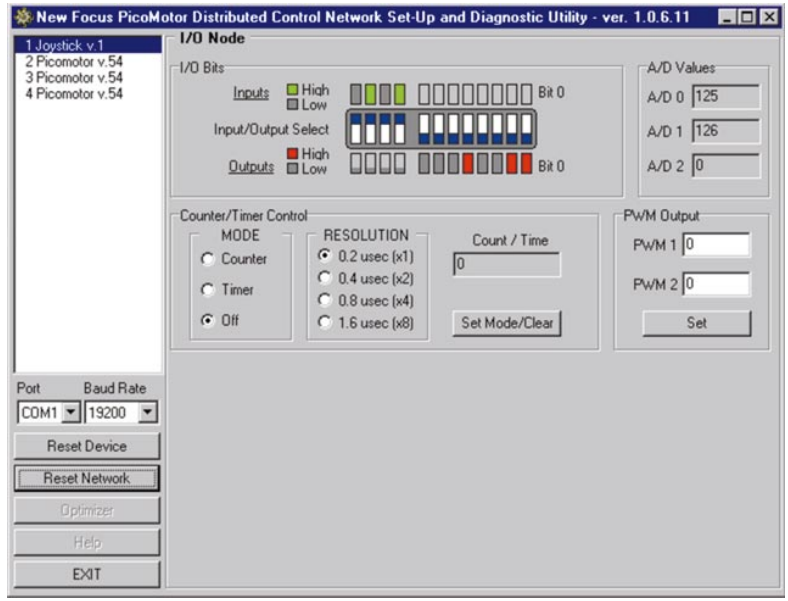
Joystick Control Panel

The Joystick Control Panel, shown in Figure 15, is displayed when the Joystick is selected from the module list displayed on the left side of the DCN Utility window.

Note: *The connectivity for this configuration menu is different than that for the configuration menu shown in Figure 14, which only included Picomotor drivers. Please refer to "Using the RS-485 Interface" on page 53 for further details.*

The various options available on the Joystick Control Panel are described in the following sections.

Figure 15: Joystick Control Panel



I/O Bits Panel

The I/O Bits panel lets the user view the state of the input bits and set the value of the output bits.

Viewing Input Bits

The row of LED indicators indicates the state of the input bit (red for set, white for clear).

Setting/Clearing Output Bits

The row of LED indicators below the input bits is used to set or clear an output bit. If the LED is white, the bit is clear (0 volts) and if the LED is red, the bit is set. Clicking on the output LED will toggle its value.

A/D Values Panel

The A/D Values panel displays the values of the two 8-bit A/D input channels that correspond to the joystick X and Y axes positions. The values displayed are between 0 and 255 with approximately 127 being the middle of the range of motion of the joystick.

Computer Control: Using DLLs

Overview

The DCN Function Library (Ldcnlib.dll) is a dynamic link library of functions that can be used to develop custom applications for the New Focus Intelligent Picomotor drivers and joystick modules. This library file, along with sample programs in LabVIEW, Visual Basic and Visual C++, can be found on the New Focus web site.

The following pages contain a summary of all available commands, followed by detailed definitions for each command, including syntax, variables, and examples (in C++). Before you begin using the library, you may also want to read the "Computer Control: Global Definitions" chapter beginning on page 101.

Note: *The DLL library cannot be used with the Model 8750 network controller.*

Using the RS-485 Interface

The host computer can communicate with the modules using an RJ-45 cable (with a DB-9/RJ-45 adapter) connected to the **Network In** connector of a joystick or driver. This cable and adapter are available in the optional New Focus Model 8761 Intelligent Picomotor Computer Interface Kit (see "Intelligent Picomotor Accessories" on page 10).

Note: *See "Using DLL/DCN with the Driver(s) and Joystick" on page 18 or "Using DLL/DCN with the Driver(s) Only" on page 17 for set-up instructions.*

Programming for the Driver(s) and Joystick

When programming for the driver(s) and joystick, keep the following rules in mind:

- The host dynamically sets the address of each module with the aid of the daisy-chained **Network In** and **Network Out** lines. The first module in the network is assigned the default address of 0x00.
- Each driver can support up to three motors.
- The joystick has four digital inputs (buttons numbered 0 to 3) and eight digital outputs (LEDs numbered 0 to 7). The analog inputs are expressed in (x,y) coordinates of the joystick axes.

Note: *See the "Computer Control: Global Definitions" chapter beginning on page 101 for additional information about addressing, byte descriptions, and more.*

Note: *All of the ".h" files referred to in the examples are available on the New Focus web site along with the C++ example.*

Conventions

The following conventions are used in both the "Command Summary" and the "Command Definitions" sections.

- The commands are case sensitive.
- Variables to be defined are shown in parantheses ().
- Response variables appear before the command.

For example, in the command

```
num_modules = LdcnFullInit(*portname, baudrate)
```

`num_modules` is the response variable, `LdcnFullInit` is the command, and `*portname` and `baudrate` are the variables to be defined. The variable data types are explained in the table on the next page.

Variable Data Types

Every command definition has an argument/parameter table and a response table that list the data types for each variable. The primary types and their ranges are described in the table below.

Type	Description	Range	C/C++	Visual BASIC
bool	Boolean	0, 1	BOOL	Boolean
u8	8-bit ASCII character	0 to 255	Char	Not supported by BASIC. For functions that require character arrays, use string types instead.
i16	16-bit signed integer	–32,768 to 32,767	Short	Integer
u16	16-bit unsigned integer	0 to 65,535	Unsigned short for 32-bit compilers.	Not supported by BASIC. For functions that require unsigned integers, use the signed integer type instead. See the i16 description.
i32	32-bit signed integer	–2,147,483,648 to 2,147,483,647	Long	Long
u32	32-bit unsigned integer	0 to 4,294,967,295	Unsigned long	Not supported by BASIC. For functions that require unsigned long integers, use the signed long integer type instead. See the i32 description.
f32	32-bit single-precision floating point	-3.402823×10^{38} to 3.402823×10^{38}	Float	Single
f64	64-bit double-precision floating point	$-1.797683134862315 \times 10^{308}$ to $1.797683134862315 \times 10^{308}$	Double	Double

Command Summary

Common Commands

Command	Description	Page
LdcnChangeBaud	Change Baud Rate	68
LdcnDefineStatus	Define Status Data	77
LdcnFullInit	Full-Initialize Modules	70
LdcnGetGroupAddr	Query Group Address	70
LdcnGetModType	Query Module Type	71
LdcnGetModVer	Query Firmware Version	77
LdcnGetStat	Get Status	72
LdcnGetStatItems	Get Status Items	73
LdcnGroupLeader	Query Group Leader	73
LdcnHardReset	Reset Modules	77
LdcnInit	Initialize Modules	74
LdcnNoOp	Issue No Operation Command	75
LdcnReadStatus	Query Status Data	76
LdcnSetGroupAddr	Set Group Address	77
LdcnShutdown	Shutdown Module	77

Picomotor Driver Commands

Command	Description	Use While Moving?	Page
ServoStartMotion	Start Motion	Yes (only in vel. mode)	78
StepGetCmdAcc	Query Command Acceleration	Yes	78
StepGetCmdPos	Query Command Position	Yes	83
StepGetCmdSpeed	Query Command Velocity	Yes	79

Command	Description	Use While Moving?	Page
StepGetCtrlMode	Query Control Mode	Yes	80
StepGetInbyte	Query Input Byte	Yes	80
StepGetIObyte	Query I/O Byte	Yes	81
StepGetMinSpeed	Query Minimum Velocity	Yes	81
StepGetMvMode	Query Motion Mode	Yes	82
StepGetOutputs	Query State of Outputs	Yes	82
StepGetPos	Query Motor Position	Yes	82
StepGetStopCtrl	Query Control Mode	Yes	83
StepLoadTraj	Load Motion Trajectory	Yes (only in vel. mode)	84
StepResetPos	Reset Position	No	86
StepSetOutputs	Set Outputs	No	90
StepSetParam	Set Motion Parameters	No	88
StepStopMotor	Stop Motor	Yes	89

Joystick Commands

Command	Descripion	Page
IoBitDirIn	Set Line to Input	90
IoBitDirOut	Set Line to Output	94
IoClrOutBit	Turn Off Output	91
IoGetADCVal	Query A/D Value	91
IoGetBitDir	Query I/O Line	94
IoGetOutputs	Query Output Value	92
IoInBitVal	Query Input Value	93
IoSetOutBit	Turn On Output	93
IoSetOutputs	Set Output Values	94

Command Definitions

Common Commands

Change Baud Rate

Syntax

```
result = LdcnChangeBaud(groupaddr, baudrate)
```

Argument/ Parameters

Name	Type	Description
groupaddr	u8	Group address of modules to be changed.
baudrate	u16	Allowed values are 9600, 19200, 38400, 57600 and 115200

Definition

Changes the baud rate of all modules with group address `groupaddr` and also changes host's baud rate.

Should include all modules. A status packet returned from this command would be at the new baud rate, so typically (unless the host's baud rate can be accurately synchronized) there should be no group leader when this command is issued.

Response

Name	Type	Description
result	bool	True (1) on success; false (0) on failure.

Define Status Data

Syntax

```
result = LdcnDefineStatus(addr, statusitems)
```

Argument/ Parameters

Name	Type	Description
addr	u8	Address of module to be defined.
statusitems	u8	Status items to be sent back. The Define Status Items Byte descriptions are on page 107 for the driver and page 109 for the joystick.

Definition

For module(s) at address `addr`, defines which status data will be sent back with each command.

Note: The Picomotor driver reports the current position of the motor multiplied by 25. The control system (host) should divide this value by 25 to obtain the actual number of steps.

Response

Name	Type	Description
result	bool	True (1) on success; false (0) on failure.

Example

For driver only: To set bits 0, 3, 5, and 6, statusitems = 0x69 or statusitems can be set to bitwise OR of the following: 0x01 (SEND_POS), 0x08 (SEND_INBYTE), 0x20 (SEND_ID), and 0x40 (SEND_OUT). The corresponding constants in parentheses are defined in the example STEPPER.H file.

For joystick only: To set bits 0, 1, and 2, statusitems = 0x07 or statusitems can be set to bitwise OR of the following: 0x01 (SEND_INPUTS), 0x20 (SEND_AD1), and 0x40 (SEND_AD2). The corresponding constants in parentheses are defined in the example I0.H file.

Full-Initialize Modules

Syntax

```
num_modules = LdcnFullInit(*portname, baudrate)
```

Argument/ Parameters

Name	Type	Description
*portname	u8	Computer port used to connect to the modules. Possible values are COM1, COM2, COM3, or COM4.
baudrate	u16	Allowed values are 9600, 19200, 38400, 57600, and 115200.

Definition

When the network is left at a baud rate different than 19200, the modules will not recognize `LdcnHardReset()` command and `LdcnInit()` will not be able to initialize the network. `LdcnFullInit()` works exactly as `LdcnInit()` sequentially at different baud rates (19200, 9600, 38400, 57600 and 115200).

Response

Name	Type	Description
num_modules	i16	The number of modules found on the network.

Example

See lines 40–54 of the C++ Example beginning on page 94.

Query Group Address

Syntax

```
groupaddr = LdcnGetGroupAddr(addr)
```

Argument/ Parameters

Name	Type	Description
addr	u8	Address of module to be defined.

Definition

Returns the group address of a particular module.

Response

Name	Type	Description
groupaddr	u8	Group address of defined module.

Query Module Type

Syntax

```
mod_type = LdcnGetModType(addr)
```

Argument/ Parameters

Name	Type	Description
addr	u8	Address of module to be defined.

Definition

Returns the module type of a particular module.

Note: This data is only valid if bit 5 of the define status items byte for the module (page 107 for the driver and page 109 for the joystick) has been set in the last `LdcnDefineStatus()` or `LdcnReadStatus` function call.

Response

Name	Type	Description
mod_type	u8	Type of module (joystick or driver).

Example

See line 60 of the C++ Example beginning on page 94.

"3" = Picomotor driver (STEPMODTYPE, as defined in the example STEPPER.H file)

"2" = Joystick module (IOMODTYPE, as defined in the example STEPPER.H file).

Query Firmware Version

Syntax `mod_version = LdcnGetModVer(addr)`

Argument/ Parameters

Name	Type	Description
addr	u8	Address of module to be defined.

Definition Returns the firmware version number of a particular module.

Note: This data is only valid if bit 5 of the define status items byte for the module (page 107 for the driver and page 109 for the joystick) has been set in the last `LdcnDefineStatus()` or `LdcnReadStatus` function call.

Response

Name	Type	Description
mod_version	u8	Firmware version of the module. It will be in the range of 50 to 59.

Example See line 61 of the C++ Example beginning on page 94.

Get Status

Syntax `status = LdcnGetStat(addr)`

Argument/ Parameters

Name	Type	Description
addr	u8	Address of module to be defined.

Definition Return the last status byte of module at address `addr`.

See page 107 for driver status byte description. The joystick status byte description can be found on page 109.

Response

Name	Type	Description
status	u8	Last status byte of the module.

Get Status Items

Syntax

```
statusitems = LdcnGetStatItems(addr)
```

Argument/ Parameters

Name	Type	Description
addr	u8	Address of module to be defined.

Definition

Returns the byte specifying the default status items to be returned in the status data packet—the last sent parameter of the LdcnDefineStatus() function. The detailed description of the individual bits of the define status items byte is on page 107.

Response

Name	Type	Description
statusitems	u8	Default status items to be returned in the status data packet.

Query Group Leader

Syntax

```
result = LdcnGroupLeader(addr)
```

Argument/ Parameters

Name	Type	Description
addr	u8	Address of module to be defined.

Definition

Answers whether specified module is a group leader.

Response

Name	Type	Description
result	bool	True (1) if specified module is group leader, false (0) if not.

Reset Modules

Syntax

```
result = LdcnHardReset()
```

Definition

Resets all modules to their power-up state. Under almost all circumstances, this is issued to a group address (0xFF) including all control modules. Cleans up the internal data structure and resets the COM port's baud rate to the default value of 19200.

Response

Name	Type	Description
result	bool	True (1) on success; false (0) on failure.

Initialize Modules

Syntax

```
num_modules = LdcnInit(*portname,
baudrate)
```

Argument/ Parameters

Name	Type	Description
*portname	u8	Computer port used to connect to the modules. Possible values are COM1, COM2, COM3, or COM4.
baudrate	u16	Allowed values are 9600, 19200, 38400, 57600, and 115200.

Description

Initializes all modules on the LDCN network with unique sequential addresses starting at 1 and establishes their device types. All modules are assigned a group address of 0xFF.

Cleans up the internal data structure and resets the COM port's baud rate to the default value of 19200.

Sends a `LdcnHardReset()` command to group address 0xFF (default), and sets baud rate for all devices.

Response

Name	Type	Description
num_modules	i16	The number of modules found on the network.

Example

See line 41 of the C++ Example beginning on page 94.

Issue No Operation Command

Syntax

```
result = LdcnNoOp(addr)
```

Argument/ Parameters

Name	Type	Description
addr	u8	Address of module to be defined.

Definition

Does nothing except cause a status packet with the currently defined status data to be returned. The status byte can then be read back with a `LdcnGetStat()` function call.

Response

Name	Type	Description
result	bool	True (1) on success; false (0) on failure.

Example

See lines 87–90 and 126–129 of the C++ Example beginning on page 94.

Query Status Data

Syntax

```
result = LdcnReadStatus(addr,
statusitems)
```

Argument/ Parameters

Name	Type	Description
addr	u8	Address of module to be defined.
statusitems	u8	Status items to be sent back. The define status items byte is described on page 107 for the driver and page 109 for the joystick.

Definition

Reads status data from a module without changing the default status data. Thus, this is a non-permanent version of the `LdcnDefineStatus()` command.

The status packet returned in response to this command will incorporate the data bytes specified, but subsequent status packets will include only the data bytes previously specified with the `LdcnDefineStatus()` command. The individual bits in the define status items byte and the method to set them are defined in the description of the `LdcnDefineStatus()` command.

Note: The Picomotor driver reports the current position of the motor multiplied by 25. The control system (host) should divide this value by 25 to obtain the actual number of steps.

Response

Name	Type	Description
result	bool	True (1) on success; false (0) on failure.

Example

See lines 59, 106–107, and 148 of the C++ Example beginning on page 94.

Set Group Address

Syntax

```
result = LdcnSetGroupAddr(addr,
groupaddr, leader)
```

Argument/ Parameters

Name	Type	Description
addr	u8	Address of module to be defined.
groupaddr	u8	Group address of a module. Valid addresses are between 0x80 and 0xFF. The initial value is 0xFF.
leader	bool	Each group address has one leader. True (1) to set specified module to be group leader, false (0) if not.

Definition

Sets the group address of a module.

Designates a group leader.

Response

Name	Type	Description
result	bool	True (1) on success; false (0) on failure.

Shutdown Module

Syntax

```
LdcnShutdown()
```

Definition

Cleans up the internal data structure, resets all LDCN modules to their power-up state by sending `LdcnHardReset()` command and closes previously opened COM port.

Example

See lines 79 and 160 of the C++ Example beginning on page 94.

Picomotor Driver Commands

Start Motion

Syntax

```
result = ServoStartMotion(groupaddr)
```

Argument/ Parameters

Name	Type	Description
groupaddr	u8	Group address of drivers to be defined.

Definition

Causes the trajectory information loaded with the most recent `StepLoadTraj()` function call to be executed. This is useful for loading several drivers with trajectory information and then starting them simultaneously with a group command. This can be a valid group address or any individual address as well.

Note: The drivers must be in velocity mode to use this command when the Picomotor is moving.

Response

Name	Type	Description
result	bool	True (1) on success; false (0) on failure.

Example

See lines 137–143 of the C++ Example beginning on page 94.

Query Command Acceleration

Syntax

```
acc = StepGetCmdAcc(addr)
```

Argument/ Parameters

Name	Type	Description
addr	u8	Address of module to be defined.

Definition

Returns the commanded acceleration used in the last `StepLoadTraj()` function call in position or velocity mode.

Response

Name	Type	Description
acc	u8	Acceleration used in last `StepLoadTraj()` command.

Query Command Position

Syntax

```
pos = StepGetCmdPos(addr)
```

Argument/ Parameters

Name	Type	Description
addr	u8	Address of module to be defined.

Definition

Returns the commanded position used in the last `StepLoadTraj()` function call in position mode.

Note: The device reports the current position of the motor multiplied by 25. The control system (host) should divide this value by 25 in order to obtain the actual number of steps.

Response

Name	Type	Description
pos	i32	Position used in last `StepLoadTraj()` command.

Query Command Velocity

Syntax

```
speed = StepGetCmdSpeed(addr)
```

Argument/ Parameters

Name	Type	Description
addr	u8	Address of module to be defined.

Definition

Returns the commanded velocity used in the last `StepLoadTraj()` function call in position or velocity mode.

Response

Name	Type	Description
speed	u8	Speed used in last `StepLoadTraj()` command.

Query Control Mode

Syntax `mode = StepGetCtrlMode(addr)`

Argument/ Parameters

Name	Type	Description
addr	u8	Address of module to be defined.

Definition Returns the mode control byte used in the last `StepSetParam()` function call.

Response

Name	Type	Description
mode	u8	Mode used in last `StepSetParam()` command.

Query Input Byte

Syntax `inbyte = StepGetInbyte(addr)`

Argument/ Parameters

Name	Type	Description
addr	u8	Address of module to be defined.

Definition Returns the input byte value of a Picomotor driver. See page 108 for driver input byte description.

Note: This data is only valid if bit 3 of the Define Status Items byte (page 107) has been set in the last `LdcnDefineStatus()` or `LdcnReadStatus()` function call.

Response

Name	Type	Description
inbyte	u8	Input byte value of Picomotor driver.

Query I/O Byte

Syntax

```
IObyte = StepGetIObyte(addr)
```

Argument/ Parameters

Name	Type	Description
addr	u8	Address of module to be defined.

Definition

Returns a byte containing the I/O state byte of a Picomotor driver. See page 108 for driver I/O state byte description.

Note: This data is only valid if bit 6 of the define status items byte (page 107) has been set in the last `LdcnDefineStatus()` or `LdcnReadStatus()` function call.

Response

Name	Type	Description
lObyte	u8	I/O state byte value of Picomotor driver.

Query Minimum Velocity

Syntax

```
minspeed = StepGetMinSpeed(addr)
```

Argument/ Parameters

Name	Type	Description
addr	u8	Address of module to be defined.

Definition

Returns the commanded minimum velocity used in the last `StepSetParam()` function call.

Response

Name	Type	Description
minspeed	u8	Minimum velocity used in last `StepSetParam()` command.

Query Motion Mode

Syntax `mode = StepGetMvMode(addr)`

Argument/ Parameters

Name	Type	Description
addr	u8	Address of module to be defined.

Definition Returns the mode control byte used in the last `StepLoadTraj()` function call.

Response

Name	Type	Description
mode	u8	Speed used in last `StepLoadTraj()` command.

Query State of Outputs

Syntax `outbyte = StepGetOutputs(addr)`

Argument/ Parameters

Name	Type	Description
addr	u8	Address of module to be defined.

Definition Returns the state of the outputs (channel select and motor type) set by the last `StepSetOutputs()` function call.

Response

Name	Type	Description
outbyte	u8	State of outputs set by last `StepSetOutputs()` command.

Query Motor Position

Syntax

```
pos = StepGetPos(addr)
```

Argument/ Parameters

Name	Type	Description
addr	u8	Address of module to be defined.

Definition

Returns the current motor position of a Picomotor driver.

Note: This data is only valid if bit 0 of the define status items byte (page 107) has been set in the last `LdcnDefineStatus()` or `LdcnReadStatus()` function call.

Note: The device reports the current position of the motor multiplied by 25. The control system (host) should divide this value by 25 in order to obtain the actual number of steps.

Response

Name	Type	Description
pos	i32	Current motor position of driver.

Example

See lines 107–110 and 149 of the C++ Example beginning on page 94.

Query Control Mode

Syntax

```
mode = StepGetStopCtrl(addr)
```

Argument/ Parameters

Name	Type	Description
addr	u8	Address of module to be defined.

Definition

Returns the mode control byte used in the last `StepStopMotor()` function call.

Response

Name	Type	Description
mode	u8	Mode control byte used in last `StepStopMotor()` command.

Load Motion Trajectory

Syntax

```
result = StepLoadTraj(addr, mode, pos,
speed, acc, steptime)
```

Argument/ Parameters

Name	Type	Description
addr	u8	Address of module to be defined.
mode	u8	Mode is the load trajectory control byte; individual bits are defined in the table below.
pos	i32	The position data (range 0x80000000 to 0x7FFFFFFF) is only used as the goal position in position profile mode. While the position may range from -0x80000000 to +0x7FFFFFFF, the goal position should not differ from the current position by more then 0x7FFFFFFF. The value sent to the device should be 25 times the desired target position. For example, if the current position is 0, in order to make 100 steps, the commanded position sent to the device should be 2500.
speed	u8	The speed data (range 1 to 250) is used as the goal velocity in velocity profile mode or as the maximum velocity in trapezoidal profile mode.
acc	u8	The acceleration data (range 1 to 255) is used in both trapezoidal and velocity profile mode.
steptime	u16	Steptime parameter should be set to 0.

The individual bits of the load trajectory control byte (mode) are defined as follows:

Bit	Weight	Description
0	1	Load position data
1	2	Load velocity data
2	4	Load acceleration data
3	8	Reserved. Set to 0
4	16	Direction (0=positive, 1= negative)
5	32	Reserved. Set to 0
6	64	Reserved. Set to 0
7	128	Start motion now

Bit 4 indicates the velocity direction and is ignored in trapezoidal profile mode. Note that if bit 7 is set, the motion will begin immediately; if it is not set, the motion will be started when `ServoStartMotion` command is executed.

Definition

Loads motion trajectory information for a Picomotor driver.

Note: 1) The velocity should be greater than minimum profile velocity (see `StepSetParam()` command).

2) In velocity mode, to change the direction of motion, a stop command must first be issued before a velocity in the opposite direction is commanded.

3) The driver must be in velocity mode to use this command when the Picomotor is moving.

Response

Name	Type	Description
result	bool	True (1) on success; false (0) on failure.

Example

Thus, to set bits 0, 1, 2, 4 and 7: mode=0x97 or mode can be set to the bitwise OR of the following: 0x01 (LOAD_POS), 0x02 (LOAD_SPEED), 0x04 (LOAD_ACC), 0x10 (STEP_REV) and 0x80 (START_NOW), as defined in the example STEPPER.H file.

See lines 96–102 and 137–142 of the C++ Example beginning on page 94.

Reset Position

Syntax

```
result = StepResetPos(addr)
```

Argument/ Parameters

Name	Type	Description
addr	u8	Address of module to be defined.

Definition

Resets the current position to 0.

Note: Do not issue this command while executing a position (trapezoidal) profile motion.

Response

Name	Type	Description
result	bool	True (1) on success; false (0) on failure.

Example

See lines 131–132 of the C++ Example beginning on page 94.

Set Outputs

Syntax

```
result = StepSetOutputs(addr, outbyte)
```

Argument/ Parameters

Name	Type	Description
addr	u8	Address of module to be defined.
outbyte	u8	Outbyte is the set output control byte; individual bits are defined in the table below.

The individual bits of the output control byte are defined as follows:

Bit	Weight	Description
0	1	OUT0
1	2	OUT1
2	4	OUT2
3	8	OUT3
4	16	OUT4
5	32	Reserved. Set to 0
6	64	Reserved. Set to 0
7	128	Reserved. Set to 0

All bits are cleared after power-up or after issuing a `LdcnHardReset()` command. *The states of OUT0-OUT4 are described in "Driver Motor Selector" on page 103.*

Definition

Sets the values for the output bits. This function is used for selecting the channel (bits 0 to 3) and the motor type (bit 4).

The motor channel and type can be changed only when the motor driver is disabled. This function should therefore be called after a `StepStopMotor()` command with bit 0 in mode control byte set to 0, which will disable the driver. After changing the selected channel and/or motor type, call `StepStopMotor()` with bit 0 in mode control byte set to 1 to enable the driver.

Response

Name	Type	Description
result	bool	True (1) on success; false (0) on failure.

Example

See lines 84–85 and 123–124 of the C++ Example beginning on page 94.

Set Motion Parameters

Syntax

```
result = StepSetParam(addr, mode, min-
speed, runcur, holdcur, thermlim, em_acc)
```

Argument/ Parameters

Name	Type	Description
addr	u8	Address of module to be defined.
mode	u8	Mode is the mode control byte; individual bits are defined in the table below.
minspeed	u8	Minspeed sets the minimum velocity.
runcur	u8	Running current; should be set to 0.
holdcur	u8	Holding current; should be set to 0.
thermlim	u8	Thermal limit; should be set to 0.
em_acc	u8	Emergency acceleration; should be set to 0.

The mode control byte's individual bits are defined as follows:

Bit	Weight	Description
0	1	Speed Factor (00_b = 8x, 01_b = 4x, 10_b = 2x, 11_b = 1x
1	2	
2	4	Reserved. Set to 1
3	8	Reserved. Set to 0
4	16	Reserved. Set to 0
5	32	Reserved. Set to 0
6	64	Reserved. Set to 0
7	128	Reserved. Set to 0

Definition

Sets control parameters and limits governing the behavior of the motor. This command must be issued before any motion can be executed. Also sets minimum velocity.

Response

Name	Type	Description
result	bool	True (1) on success; false (0) on failure.

Example

To set the speed factor to 2x, mode = 0x06, or mode can be set to the bitwise OR of the following: 0x02 (SPEED_1X), and 0x04 (IGNORE_LIMITS), as defined in the example STEPPER.H file.

See lines 68–79 of the C++ Example beginning on page 94.

Stop Motor

Syntax

```
result = StepStopMotor(addr, mode)
```

Argument/ Parameters

Name	Type	Description
addr	u8	Address of module to be defined.
mode	u8	Mode is the stop control byte; individual bits are defined in the table below.

The stop control byte's individual bits are defined as follows:

Bit	Weight	Description
0	1	Turn motor on/off
1	2	Reserved.
2	4	Stop abruptly
3	8	Stop smoothly
4	16	Reserved. Set to 0
5	32	Reserved. Set to 0
6	64	Reserved. Set to 0
7	128	Reserved. Set to 0

If bit 0 of the stop control byte is set, the motor driver will be turned on (enabled). If bit 0 is cleared motor driver will be turned off (disabled), regardless of the state of the other bits. If bit 2 is set, the motor will stop moving abruptly. Setting bit 3 enters a more graceful stop mode—the motor will decelerate to a stop. Only one of bits 2 or 3 should be set at one time.

Definition

Stops the motor in the manner specified by mode and enables/disables the driver.

Response

Name	Type	Description
result	bool	True (1) on success; false (0) on failure.

Example

To set bits 0 and 3, mode=0x09, or mode can be set to the bit-wise OR of the following: 0x01 (STP_ENABLE_AMP) and 0x08 (STOP_SMOOTH), as defined in the example STEPPER.H file.

See lines 68–79 of the C++ Example beginning on page 94.

Joystick Commands

Set Line to Input

Syntax

```
result = IoBitDirIn(addr, bitnum)
```

Argument/ Parameters

Name	Type	Description
addr	u8	Address of module to be defined.
bitnum	i16	The I/O line to be used as an input.

Description

Sets the specified I/O line (bitnum) to be used as an input.

Response

Name	Type	Description
result	bool	True (1) on success; false (0) on failure.

Set Line to Output

Syntax

```
result = IoBitDirOut(addr, bitnum)
```

Argument/ Parameters

Name	Type	Description
addr	u8	Address of module to be defined.
bitnum	i16	The I/O line to be used as an output.

Description

Sets the specified I/O line (bitnum) to be used as an output.

Response

Name	Type	Description
result	bool	True (1) on success; false (0) on failure.

Turn Off Output

Syntax

```
result = IoClrOutBit(addr, bitnum)
```

Argument/ Parameters

Name	Type	Description
addr	u8	Address of module to be defined.
bitnum	i16	The output bit to be cleared. Valid values are 0 to 7.

Description

Clears the value of output bit bitnum to 0 (turns off a joystick LED).

Response

Name	Type	Description
result	bool	True (1) on success; false (0) on failure.

Query A/D Value

Syntax

```
ADCVal = IoGetADCVal(addr, channel)
```

Argument/ Parameters

Name	Type	Description
addr	u8	Address of module to be defined.
channel	i16	The motor channel to be defined.

Definition

Returns the A/D value (joystick axes coordinates) from channel 0 or 1.

Note: This data is only valid if bit 1 or 2 of the define status items byte (page 109) has been set in the last `LdcnDefineStatus()` or `LdcnReadStatus()` function call.

Response

Name	Type	Description
ADCVal	u8	The joystick axes positions of the defined motor channel.

Query I/O Line

Syntax

```
result = IoGetBitDir(addr, bitnum)
```

Argument/ Parameters

Name	Type	Description
addr	u8	Address of module to be defined.
bitnum	i16	The I/O line to be defined.

Description

Returns whether the specified I/O line (bitnum) is an input.

Response

Name	Type	Description
result	bool	True (1) if specified I/O line is an input; false (0) if it is not an input.

Query Output Value

Syntax

```
outbyte = IoGetOutputs(addr)
```

Argument/ Parameters

Name	Type	Description
addr	u8	Address of module to be defined.

Description

Returns the most recently set state of an output byte (joystick LEDs).

Response

Name	Type	Description
outbyte	u16	Most recently set state of joystick LEDs.

Query Input Value

Syntax

```
result = IoInBitVal(addr, bitnum)
```

Argument/ Parameters

Name	Type	Description
addr	u8	Address of module to be defined.
bitnum	i16	The joystick button to be defined. Valid values are 1 to 4.

Definition

Returns the value of input bit bitnum (joystick buttons).

Note: This data is only valid if bit 0 of the define status items byte (page 109) has been set in the last `LdcnDefineStatus()` or `LdcnReadStatus()` function call.

Response

Name	Type	Description
result	bool	True (1) if input is on; false (0) if off.

Turn On Output

Syntax

```
result = IoSetOutBit(addr, bitnum)
```

Argument/ Parameters

Name	Type	Description
addr	u8	Address of module to be defined.
bitnum	i16	The joystick LED to be turned on. Valid values are 1 to 7.

Description

Sets the value of an I/O node output bit bitnum to 1 (turns on a joystick LED).

Response

Name	Type	Description
result	bool	True (1) on success; false (0) on failure.

Set Output Values

Syntax

```
result = IoSetOutputs(addr, outval)
```

Argument/ Parameters

Name	Type	Description
addr	u8	Address of module to be defined.
outval	u16	The values for the joystick LEDs.

Description

Sets the values for the output bits (joystick LEDs).

Response

Name	Type	Description
result	bool	True (1) on success; false (0) on failure.

Example

The following C++ code section example lists the sequence of function calls to do the following:

1. Drive a Standard Picomotor connected to channel A of the first Picomotor driver in the network, 2000 steps clockwise (forward) in position mode at a speed of 2000 Hz with an acceleration of 255.
2. Drive a Tiny Picomotor connected to channel B of the same driver counterclockwise (backward) at a speed of 1000 Hz in velocity mode with an acceleration of 225 until relative number of steps moved is approximately -3000.

The current position is continuously displayed in real-time.

```
1    //Start of program
2
3    #include "stdafx.h"
4    #include "ldcncom.h"
5    #include "stepper.h"
6    #include "servo.h"
7
```

```
8    #define  STEPMODTYPE     3
9    #define  LOAD_TRAJ       0x04 //Load trajectory data
10   #define  STOP_MOTOR      0x07 //Stop motor
11   #define  SEND_ID         0x20
12   #define  SPEED_8X        0x00 //use 8x timing
13   #define  IGNORE_LIMITS   0x04 //Do not stop automatically
14                                 //on limit switches
15   #define  POWER_ON        0x08 //set when motor power is on
16   #define  STOP_SMOOTH     0x08 //set to decelerate
17                                 //motor smoothly
18   #define  STP_ENABLE_AMP  0x01 //raise amp enable output
19   #define  STP_DISABLE_AMP 0x00 //lower amp enable output
20   #define  STP_AMP_ENABLED 0x04 //set if amplifier is
21                                 //enabled
22   #define  START_NOW       0x80
23   #define  LOAD_SPEED      0x02
24   #define  LOAD_ACC        0x04
25   #define  LOAD_POS        0x01
26   #define  SEND_POS        0x01
27   #define  STEP_REV        0x10  //reverse dir
28   #define  TYPE_TINY       0x10
29   #define  TYPE_STD        0x00
30   #define  SET_CH_A        0x00
31   #define  SET_CH_B        0x01
32
33   int main(int argc, char* argv[]) {
34   int num_modules;
35   int addr;
36   byte mod_type, mod_version;
37   byte pico_addr, outval;
38   byte mode, min_speed, run_current, hld_current, ADLimit,
em_acc, speed, acc;
39   long pos;
40
41      num_modules = LdcnInit("COM1", 19200);
```

```
42
43   // if the network is set to != 19200 baudrate
44   // the devices will not "hear" HardReset command
45   // LdcnFullInit() sends HardReset command at all
46   // possible baudrates
47
48      if (!num_modules)
49      num_modules = LdcnFullInit("COM1", 19200);
50
51      if (!num_modules) {
52     printf("No Modules found at COM1\n");
53     return 1;
54      }
55
56      // look for pico motor drivers
57      pico_addr = 0;
58      for (addr = 1; addr <= num_modules; addr++) {
59          LdcnReadStatus(addr, SEND_ID);
60          mod_type = LdcnGetModType(addr);
61          mod_version = LdcnGetModVer(addr);
62
63         if ((mod_type == STEPMODTYPE) && (mod_version >= 50)
&& (mod_version < 60))
64     { pico_addr = addr; break; }
65      }
66
67      if (pico_addr) {
68   // set parameters ----------------------------------------
69
70         min_speed = 1;
71     run_current = 0;
72     hld_current = 0;
73     ADLimit = 0;
74     em_acc = 255;
75         mode = SPEED_8X;     // or mode = SPEED_2X or
```

```
76                              // mode = SPEED_4X
77          mode |= IGNORE_LIMITS;
78          if (!StepSetParam(pico_addr, mode, min_speed,
run_current, hld_current, ADLimit, em_acc))
79      { printf("Communication Error");  LdcnShutdown(); return 1; }
80
81    // Select Motor Type and Channel ------------------------
82      outval = TYPE_STD | SET_CH_A;
83
84    // Send output value to the device
85      StepSetOutputs(pico_addr, outval);
86
87    // Read device status
88      LdcnNoOp(pico_addr);
89          if (LdcnGetStat(pico_addr) & POWER_ON == 0)
90      printf("Invalid channel");
91
92    // Enable Driver ----------------------------------------
93      StepStopMotor(pico_addr, STOP_SMOOTH | STP_ENABLE_AMP);
94
95    // Load Trajectory ----------------------------------------
-
96    // Position mode (Velocity mode: mode = START_NOW |
97    // LOAD_SPEED | LOAD_ACC;)
98          mode = START_NOW | LOAD_SPEED | LOAD_ACC | LOAD_POS;
99          pos = 25*2000;//2000 steps
100         speed = 250;//2000 Hz
101         acc = 255;//max. acc.
102   StepLoadTraj(pico_addr, mode, pos, speed, acc, 0);
103
104   // wait end of the motion
105           do {
106      LdcnReadStatus(pico_addr, SEND_POS);
107      // read device status and current position
108      pos = StepGetPos(pico_addr)/25;//read steps
```

```
109     printf("   Position: %d\n", pos);
110  } while (LdcnGetStat(pico_addr) & MOTOR_MOVING);
111
112  // Disable driver amp (STOP_ABRUPT can also be used
113  // instead of STOP_SMOOTH)
114    StepStopMotor(pico_addr, STOP_SMOOTH);
115
116  // Wait 2 secs.
117  Sleep(2000);
118
119  // Drive different motor---------------------------------
120  // Select new Motor Type and new
121  // Channel (Tiny Type, Channel B)
122  outval = TYPE_TINY | SET_CH_B;
123        StepSetOutputs(pico_addr, outval);
124        // send output value to the device
125
126  // Read device status
127    LdcnNoOp(pico_addr);
128        if (LdcnGetStat(pico_addr) & POWER_ON == 0)
129    printf("Invalid channel");
130
131  // Reset Position
132  StepResetPos(pico_addr);
133
134  // Enable Driver Amplifier
135    StepStopMotor(pico_addr, STOP_SMOOTH | STP_ENABLE_AMP);
136
137  //Reload speed, mode (switch to velocity mode,
138  //reverse direction, start with ServoStartMotion command)
139  speed = 125;//1000 Hz
140  acc = 225;//Lower acc.
141  mode = LOAD_SPEED | LOAD_ACC | STEP_REV;
142  StepLoadTraj(pico_addr, mode, pos, speed, acc, 0);
143  ServoStartMotion(pico_addr);//Start motion
```

```
144
145  // Wait for end of motion, Read device status
146  //and current position
147  do {
148  LdcnReadStatus(pico_addr, SEND_POS);
149  pos = StepGetPos(pico_addr)/25;
150  printf("Position: %d\n", pos);
151  } while (pos>=-3000);//Move approximately 3000 steps
152
153  //Stop Motor Abruptly
154    StepStopMotor(pico_addr, STOP_ABRUPT | STP_ENABLE_AMP);
155
156  // Disable driver amp
157    StepStopMotor(pico_addr, STOP_ABRUPT);
158     }
159
160       LdcnShutdown();
161
162  return 0;
163  }
164
```

Computer Control: Global Definitions

Addressing

Dynamic Addressing

Rather than using the hard-wired or switch-selected address of each DCN node, the host dynamically sets the address of each node with the aid of the daisy-chained **Network In** and **Network Out** lines. This allows additional DCN nodes to be added to an RS-485 network with no hardware changes.

On power-up, **Network In** of the first DCN node is pulled low, its communication is enabled, and the default address is 0x00. When a command is issued to give this node a new unique address, it will lower its **Network Out** line. Connecting **Network Out** to the **Network In** of the next node on the network will enable its communication at the default address of 0x00. Repeating this procedure allows a variable number of controllers present to be given unique addresses.

Group Addresses

In addition to the individual address, each node has a secondary group address. Several DCN nodes may share a common group address. This address is useful for sending commands which must be performed simultaneously by a number of nodes (e.g., `LdcnChangeBaud( )`, etc.).

When a driver or joystick receives a command sent to its group address, it will execute the command but not send back a status packet. This prevents data collisions on the shared response line. When programming group addresses, however, the host can specify that one

member of the group is the "group leader." The group leader will send back a status packet just like it would for a command sent to its individual address.

The group address is programmed using the `LdcnSetGroupAddr()` command.

Intelligent Picomotor Driver

Driver Identification

After power-up or `LdcnHardReset()` command and before first `StepStopMotor()` command with bit 0 set, input bits IN0 to IN5 from the input byte are used to identify the device type.

For the Model 8753, the identification number is 0x01. The identification sequence should occur after initializing the network and reading the device type and version:

1. Read the states of input bits IN0 to IN5.
2. Set OUT4 to 1.
3. Read the states of input bits IN0 to IN5.

If the input states are inverted (see table below), the device's identification number is the value in Step 1.

OUT4	IN5	IN4	IN3	IN2	IN1	IN0
0	0	0	0	0	0	1
1	1	1	1	1	1	0

Note: *The identification number is valid until first set to clear transition of OUT 4 or before the first Stop Motor command with bit 0 set.*

An example of an identification sequence is as follows:

```
// C++ code segment to verify that a module at address pico_addr
is a Picomotor Driver.

LdcnReadStatus(pico_addr, SEND_INBYTE);
id1=StepGetInByte(pico_addr);
```

```
StepSetOutputs(pico_addr, 0x10);
LdcnReadStatus(pico_addr, SEND_INBYTE);
id2=StepGetInByte(pico_addr);
StepSetOutputs(pico_addr, 0);
id1 &= 0x3F; //six bits ID
id2 |= 0xC0;
if ((id1==~id2) && (id1==1)) {
//This is a Picomotor drive
}
```

Driver Motor Selector

The Model 8753 has five internal control signals, OUT0 to OUT4, used to control the motor connector and type selector. OUT0 to OUT2 selects the motor connector. OUT3 is reserved and should be cleared. OUT4 selects between the driver signal for the Standard Picomotor or Tiny Picomotor. The current motor type and connector selection can be changed after sending `StepStopMotor()` command with bit 0 in the stop control byte cleared. If the selected motor connector is not supported, bit 3 in the status byte will be cleared. The sequence to select another channel and/or change the motor type is as follows:

1. `StepStopMotor()` command with bit 0 cleared (stop motion and disable motor driver).
2. `StepSetOutputs()` command with desired motor connector (see table below).

OUT4	OUT3	OUT2	OUT1	OUT0	Status byte bit 3	Motor Selected
0	0	0	0	0	1	MOTOR A Standard
0	0	0	0	1	1	MOTOR B Standard
0	0	0	1	0	1	MOTOR C Standard
1	0	0	0	0	1	MOTOR A Tiny
1	0	0	0	1	1	MOTOR B Tiny
1	0	0	1	0	1	MOTOR C Tiny

OUT4	OUT3	OUT2	OUT1	OUT0	Status byte bit 3	Motor Selected
X	X	X	1	1	0	N/A
X	X	1	X	X	0	N/A
X	1	X	X	X	0	N/A

3. `StepStopMotor()` command with bit 0 of the stop control byte set will enable the selected motor if bit 3 = 1.

A Picomotor can be damaged if it is driven with the wrong type of driver signal for an extended period of time, so it is important to ensure that the motor driver is configured to generate the correct driver signals.

Driver Diagnostic

The Model 8753 is protected against motor output short and overtemperature. In addition, a missing motor can be detected while moving in negative direction. Status byte bit 2 (Motor On) and IN0 (STOP), IN1, and IN2 form the input byte that is used for diagnostic.

Motor On	IN2	IN1	IN0 (STOP)	Diagnostic
X	0	0	0	OK
0	0	0	1	NO MOTOR (single step negative direction only with OUT4= 0)
0	1	0	1	MOTOR SHORT
X	X	1	0	OVER TEMPERATURE
0	X	1	1	OVER TEMPERATURE latched

Velocity Profile Mode

Velocity profile mode is used to smoothly accelerate from one velocity to another. Commanded velocities are specified as integer values between 1 and 250. Minimum and maximum velocities for the different speed modes appear in the table below:

Speed factor	Step Rate Multiplier (K) (Minimum Velocity) in steps/sec.	Max. Step Rate in steps/sec. (Velocity = 250)
1x	1	250
2x	2	500
4x	4	1,000
8x	8	2,000

Step Rate Multipliers and Maximum Velocities for Different Speed Factors

The actual velocity V in steps per second can be obtained using the formula $V = S * K$, where

S is commanded velocity (range 1 to 250) and

K is the step rate multiplier for current speed factor (see the table)

The acceleration or deceleration is achieved by incrementing (or decrementing) the current integer velocity value by one until the goal velocity is reached. The actual time for acceleration from one velocity to another can be obtained using the formula:

$T_{acc} = | (64 - 0.25 * Acc) * (S1 - S0) |$ in ms, where

Acc is the acceleration value (range 1–255),

S0 is current velocity (range 1–250),

S1 is target velocity (range 1–250), and

T_{acc} is the time to accelerate from velocity *S0* to *S1* with acceleration *Acc*

Examples

1. Accelerating to velocity 125 with minimum profile velocity = 25 and acceleration = 100.

Acc = 100, S0 = 25, S1 = 125.

| (64 – 0.25 * 100) * (125 – 25) | = | 39 * 100 | = | 3900 | = 3900 ms (3.9 s)

2. Decelerating from velocity 125 to stop with minimum profile velocity = 25 and acceleration 100.

Acc = 100, S0 = 125, S1 = 25.

| (64 – 0.25 * 100) * (25 – 125) | = | 39 * (- 100) | = | -3900 | = 3900 ms (3.9 s)

Note: *To change the direction of motion, a stop command must first be issued before a velocity in the opposite direction is commanded.*

Driver Power-Up and Reset Conditions

On power-up or reset, the following state is established:

- Motor position is reset to zero.
- Velocity and acceleration values are set to zero.
- All parameters are set to zero.
- All outputs are cleared.
- The motor driver is disabled.
- The default status data is the status byte only.
- The individual address is set to 0x00.
- The group address is set to 0xFF (group leader is not set).
- Communications are enabled or disabled depending on "A in."
- "A out" is HIGH, disabling the next drive communications.
- The baud rate is set to 19.2 Kbps.

Driver: Define Status Items Byte

Default = 0x00

Bit	Description	Bytes Sent
0	Send position	4 bytes
1	Reserved. Set to 0	N/A
2	Reserved. Set to 0	N/A
3	Send input byte	byte
4	Reserved. Set to 0	N/A
5	Send device ID and version number	2 bytes (driver device ID = 3, version number = 50–59)
6	Send I/O state	byte
7	Reserved. Set to 0	N/A

Driver: Status Byte

Bit	Name	Definition
0	Motor is moving	This bit is set when the motor is moving and cleared otherwise.
1	Checksum error	Set if there was a checksum error in the command packet received.
2	Motor is on.	Set if the motor power driver is enabled
3	Motor selector status	Cleared if selected motor connector is out of range.
4	At commanded velocity	Set if the commanded velocity is reached.
5	Velocity profile mode	Set if the motor is moving in velocity mode.
6	Position (trapezoidal) profile mode	Set if the motor is moving in trapezoidal mode.
7	Reserved	

Driver: Input Byte

Bit	Name	Definition
0	IN0	The value of bit IN0 (STOP)
1	IN1	The value of bit IN1
2	IN2	The value of bit IN2
3	IN3	The value of bit IN3
4	IN4	The value of bit IN4
5	IN5	The value of bit IN5
6	Reserved	
7	Reserved	

Driver: I/O State Byte

Bit	Name	Definition
0	IN0	The value of bit IN0 (STOP)
1	IN1	The value of bit IN1
2	IN2	The value of bit IN2
3	OUT0	The value of Motor selector bit 0
4	OUT1	The value of Motor selector bit 1
5	OUT2	The value of Motor selector bit 2
6	OUT3	The value of output bit 3 (Reserved)
7	OUT4	The value of Motor selector bit 4 (0 = Standard, 1 = Tiny)

Note: IN0, IN1 and IN2 in input byte and I/O state byte are the same inputs.

Joystick

Joystick: Define Status Items Byte

Default = 0x00

Bit	Description	Bytes Sent
0	Send I/O Byte 0 and Byte 1	2 bytes (button 1is bit 0 in I/O byte 1, button 2 is bit 1 in I/O byte 1, and so on)
1	Send ANALOG IN 0 value—Joystick X axis	1 byte
2	Send ANALOG IN 1 value—Joystick Y axis	1 byte
3	Reserved	N/A
4	Send timer value	4 bytes, least significant first
5	Send device ID, version number	2 bytes (8754 device ID = 2, version number = 1)
6	Send I/O bit values captured with the Synch Input command	2 bytes
7	Send timer value captured with the Synch Input command.	4 bytes

Joystick: Status Byte

Bit	Description
0	Undefined
1	Checksum error detected (if set)
2	Undefined
3	Undefined
4	Undefined
5	Undefined
6	Undefined
7	Undefined

Troubleshooting

Joystick

Moving the Joystick Does Not Move Any Picomotors

When moving the joystick does not move any Picomotors, it could indicate any of the following:

- **The Picomotor is at the end of its travel range**.
- **A motor is not connected to the selected axis**. Check the connections, or switch to another axis.
- **The cables are not connected or are loose.** Check the connections.
- **The joystick is disabled.** If all three driver LEDs are on, the joystick has been manually disabled. Press the **Set Axis/Enable** button and then the **X+Y/Enable** button on top of the joystick to enable it for use. If all of the LEDs are on, the joystick has been set to Command Mode from computer control. Issue a `JON` command from MCL to enable the joystick and return to stand-alone mode.
- **Both joystick axes are disabled**. If two driver LEDs are on but no motor LEDs are lit, then no motors have been assigned. See "Selecting the Motors to Control" on page 24 for instructions.

All LEDs on Joystick are On

The network controller is set to command mode, and the joystick is disabled. Use the `JON` command from MCL to switch to stand-alone mode and enable the joystick.

No LEDs on Joystick are On

If no LEDs are lit on the joystick:

- **The joystick may not be connected to the network.** Check the cable connections.
- **The joystick may not have power.** Verify that the joystick dip switch settings are in the default position (page 126). If the dip switch settings are correct, refer to the "Setting Up" chapter beginning on page 13 to check that your set-up is correct.

Three Driver LEDs on Joystick are On

The joystick is disabled. Press the **Set Axis/Enable** button and then the **X+Y/Enable** button to enable the joystick.

Motor Light on Joystick Will Not Illuminate

The motor is already programmed for the other axis, or the motor is not selected. Press the **Motor** button until the desired motor is selected.

Driver LED on Joystick Cannot Be Selected

The Picomotor driver was not present during boot-up. Connect the driver and reboot. (This can be done either with a power-up or by pressing the **Set Axis/Enable**, **Driver**, and **Motor** buttons on the joystick.)

Set X and Set Y LEDs Flash on Joystick

No drivers are detected or present on the network. Check your connections and reboot.

Joystick Axes Settings Return to Default After Power-Up

The battery backup for the RAM is not enabled on the network controller. Verify that SW5 on the controller is on (the switch setting should be up).

Network Controller

Network Controller LED Flashes

The joystick is not connected to the network. Check the cable connections.

Speed Values Return to Default After Power-up

The changes to the speed values were not saved before shutdown. After using MCL commands to change speed values, issue a *SAV* command to retain the values.

MCL Commands Not Functioning Correctly

If the `REL`, `FOR`, and `REV` commands are not functioning as desired, try sending a `JOF` command first.

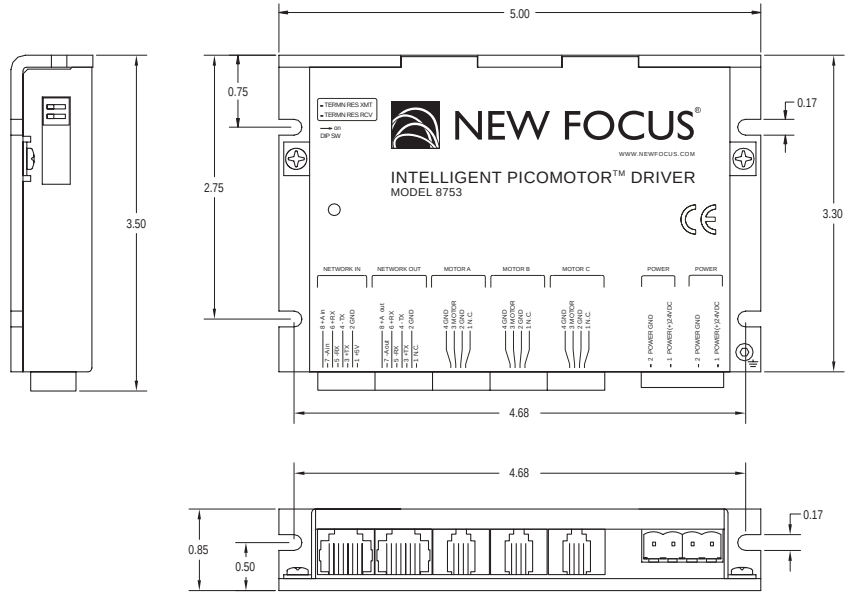
No Line Feeds When Using MCL in Hyperterminal

You may experience problems with line feeds in some versions of Hyperterminal. An alternate program, Network Controller Terminal, is available on the New Focus web site.

Specifications

Intelligent Picomotor Drivers

Driver Mechanical Drawings



Note: All dimensions are in inches.

Driver Characteristics

Specification	Model 8753
Power Supply Voltage	21 to 29 VDC
Power Supply Current	Average 0.65 Amp DC (typical)

Specification	Model 8753
Output Frequency Range	1 Hz to 2KHz
Maximum Output Frequency (w/o forced cooling	1.0 kHz @ 100% duty cycle 2 kHz @ 50% duty cycle (ON time max 120 sec)
Number of Channels	3
Number of Active Channels at Once	One
Communication Protocol	Distributed Control Network (DCN)
Maximum Number of Drivers (per network)	31
Communication Interface	RS-485
Serial Baud Rate	19.2 to 115.2 Kbps
LED (two intensity levels)	Power 'OK'—low intensity Ready—high intensity
Output Short Protection (motor output to ground)	Shutdown if motor output is shorted
Over Temperature Protection	Shutdown at 70˚ C
Fire Safety: Internal Fuse	3A Quick blow
Storage Temperature	–30 to +85˚ C
Operating Temperature	10 to 45˚ C
Power Supply Connectors	2 x 2-pin Phoenix MSTB 2.5/2-ST-5.08 (or equivalent)
Picomotor Connectors	4-pin RJ-22 (3x)
Communication Interface	8-pin RJ-45
Size (L x H x D)	5" x 0.85" x 3.3"
Weight	0.55 lb (250 g)
Certification	CE In conformity with the following standards: UL 60950:2000, EN 60950:2000, CSA E60065-00, 89/336/EEC

Note: Rated at 25 ºC ambient, POWER (+) = 24VDC

Driver Pinouts

Driver DIP Switches

SW	Signal	Description
1	Term Res RCV	Receive line terminator (Default=off. The last driver in the network should have this on.)
2	Term Res XMT	Transmit line terminator (Default=off. The last driver in the network should have this on.)

Driver Power

Pin	Signal	Description
1	POWER (+) 24 VDC	+21 to +29 VDC power supply, positive terminal
2	POWER GND*	Power supply ground
1	POWER (+) 24 VDC	+21 to +29 VDC power supply, positive terminal
2	POWER GND*	Power supply ground

* POWER GND and GND are electrically connected. Driver's case is isolated from the circuitry and can be grounded externally.

Driver Motor A, Motor B, and Motor C Connectors

Pin	Signal	Description
1	N.C.	Not connected
2	GND*	Power ground
3	MOTOR	Motor output
4	GND*	Motor ground

* POWER GND and GND are electrically connected. Driver's case is isolated from the circuitry and can be grounded externally.

Driver Network Out

Pin	Signal	Description
1	N.C.	Not connected
2	GND*	Interface ground
3	+TX	(+) Transmit data
4	–TX	(–) Transmit data
5	–RX	(–) Receive data
6	+RX	(+) Receive data
7	–A out	(–) Address output
8	+A out	(+) Address output

* POWER GND and GND are electrically connected. Driver's case is isolated from the circuitry and can be grounded externally.

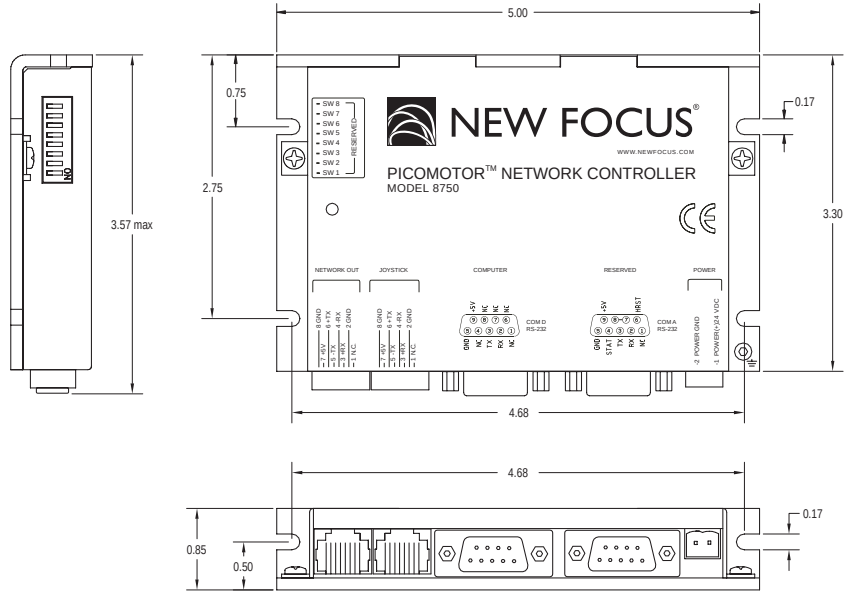
Driver Network In

Pin	Signal	Description
1	+5V	RS-232 adapter power supply (200 mA Max)
2	GND*	Interface ground
3	+TX	(+) Transmit data
4	–TX	(–) Transmit data
5	–RX	(–) Receive data
6	+RX	(+) Receive data
7	–A in	(–) Address input
8	+A in	(+) Address input

* POWER GND and GND are electrically connected. Driver's case is isolated from the circuitry and can be grounded externally.

Intelligent Picomotor Network Controller

Network Controller Mechanical Drawings



Note: All dimensions are in inches.

Network Controller Characteristics

Specification	Model 8750
Power Supply Voltage	12 to 32 VDC, (10 to 40 VDC Abs. Max range); Supply current <100 mA at 24 VDC
CPU	Rabbit 2000™—18.432 MHz
Flash Memory	256K
RAM	128K
RAM Backup Battery	3V—CR2032
COM A	RS-232, D-sub 9-pin connector (female)
Com D	RS-232 or RS-485 half duplex (2 wire), D-sub 9-pin connector (female)
Network Out	RS-485 full duplex (4 wire) DCN-compatible, 8-pin RJ-45 connector
Joystick	RS-485 full duplex (4 wire) DCN-compatible, 8-pin RJ-45 connector
Power Connector	Phoenix MSTB 2.5/2-ST-5.08 (or equivalent)
Storage Temperature	–30 to +85˚ C
Operating Temperature	0 to 45˚ C
Size	5.00" x 3.30" x 0.85"
Weight	0.55 lb (250 g)
Certification	CE In conformity with the following standards: EN 60950:2000, 89/336/EEC

Note: Rated at 25˚C ambient, POWER(+) 24VDC

Network Controller Pinouts

Network Controller DIP Switches

SW	Function	Description	Factory Default
1	SWITCH A (/PD0)	Configuration switch connected to PD0 (ON = logic "0")	0
2	SWITCH B (/PD1)	Configuration switch connected to PD1 (ON = logic "0")	0
3	SWITCH C (/PD2)	Configuration switch connected to PD2 (ON = logic "0")	0
4	SWITCH D (/PD3)	Configuration switch connected to PD3 (ON = logic "0")	0
5	BATTERY ON/OFF	RAM backup battery ON/OFF	1
6	COM A COLD BOOT	ON = COM A COLD BOOT ENABLED	0
7	HOST RESET EN	ON = HOST RESET ENABLED	0
8	CPU RESET SW	ON = CPU RESET	0

Network Controller Power Connector

Pin	Signal	Description
1	POWER (+) 24V	12 to 32 V power supply, positive terminal
2	POWER GND*	Power supply ground

Network Controller COM A Connector

Pin	Signal	Description
1	N.C.	Not connected
2	RX	Receive data
3	TX	Transmit data
4	STAT	STATUS output from Rabbit 2000 CPU (used by software development tools)

Pin	Signal	Description
5	GND*	Interface ground
6	HRST	HOST RESET input (used by software development tools) Enabled by HOST RESET EN switch
7	Connected to pin 8	
8	Connected to pin 7	
9	+5V**	+5V Power output

* POWER GND and GND are electrically connected. Controller's case is isolated from the circuitry and can be grounded externally.

** 250-mA maximum for all outputs combined.

Network Controller COM D

Pin	Signal	Description
1	N.C.	Not connected
2	RX	Receive data
3	TX	Transmit data
4	N.C.	Not connected
5	GND*	Interface ground
6	N.C.	Not connected
7	N.C.	Not connected
8	N.C.	Not connected
9	+5V**	+5V Power output

* POWER GND and GND are electrically connected. Controller's case is isolated from the circuitry and can be grounded externally.

** 250 mA MAX for all outputs combined.

Network Controller Joystick Connector

Pin	Signal	Description
1	N.C.	Not Connected
2	GND*	Interface ground
3	+RX	(+) Receive data
4	–RX	(–) Receive data
5	–TX	(–) Transmit data
6	+TX	(+) Transmit data
7	+5V**	+5V Power output
8	GND*	Interface ground

* POWER GND and GND are electrically connected. Controller's case is isolated from the circuitry and can be grounded externally.

** 250 mA MAX for all outputs combined.

Network Controller Network Out Connector

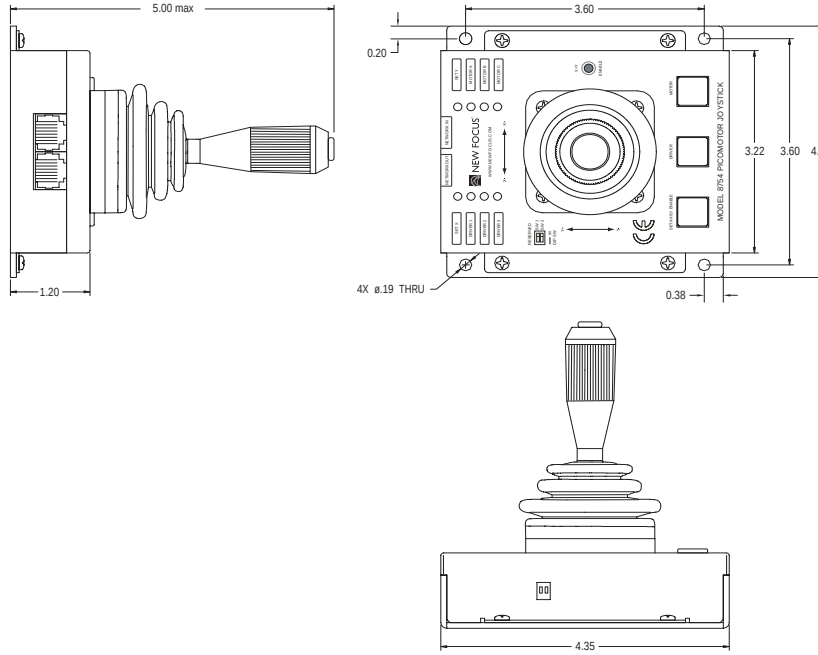
Pin	Signal	Description
1	N.C.	Not Connected
2	GND*	Interface ground
3	+RX	(+) Receive data
4	–RX	(–) Receive data
5	–TX	(–) Transmit data
6	+TX	(+) Transmit data
7	+5V**	+5-V Power output
8	GND*	Interface ground

* POWER GND and GND are electrically connected. Controller's case is isolated from the circuitry and can be grounded externally.

** 250 mA MAX for all outputs combined.

Joystick

Joystick Mechanical Drawings



Note: All dimensions are in inches.

Joystick Characteristics

Specification	Model 8754
Power Supply Voltage	4.75 to 5.25 VDC
Power Consumption	<50 mA
Number of Axes	2
Axes Resolution	8 bits
Power Supply and Communication Interface	8-pin RJ-45
Storage Temperature	–30 to +85˚ C
Operating Temperature	0 to 45˚ C
Size (L x H x D)	4.35" x 4.00" x 4.90"
Weight	0.6 lb (0.260 kg)
Certification	CE In conformity with the following standards: EN 60950:2000, 89/336/EEC

Note: Rated at 25˚ C ambient.

Joystick Pinouts

Joystick DIP Switches

SW	Description
1	Power is supplied from **Network Out**. (Default is off)
2	Power is supplied from **Network In.** (Default is on)

Joystick Network Out

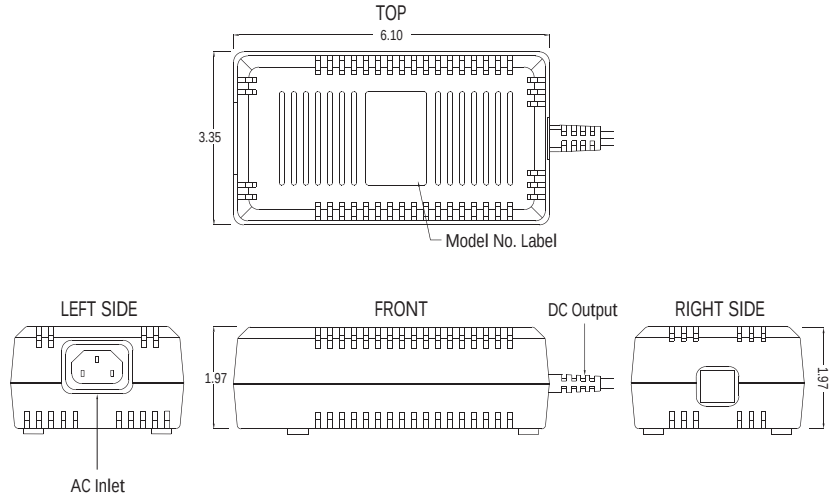
Pin	Signal	Description
1	+5 V slave	**SW1**=on: +5 V power supply from slave **SW1**=off: Not connected
2	GND	Interface ground
3	+TX	(+) Transmit data
4	–TX	(–) Transmit data
5	–RX	(–) Receive data
6	+RX	(+) Receive data
7	–A out	(–) Address output
8	+A out	(+) Address output

Joystick Network In

Pin	Signal	Description
1	+5V	**SW1**=on: +5-V power supply from slave **SW1**=off: Not connected
2	GND	Interface ground
3	+TX	(+) Transmit data
4	–TX	(–) Transmit data
5	–RX	(–) Receive data
6	+RX	(+) Receive data
7	+5V host	**SW1**=off and **SW2**=on: +5 V power supply from host
8	+A in	(+) Address input

Driver Kit Power Supply

Power Supply Mechanical Drawings



Note: All dimensions are in inches.

Power Supply Characteristics

Specification	Model 8755
Input Range	90–264 VAC (wide range)
Frequency	47–63 Hz
Input Current (rms)	2 A @ 90 VAC; 1 A @ 264 VAC Max.
Efficiency	>80% @ full load, 115 VAC (DC conversion)
EMI/RFI	FCC Part 15, Subpart B Class B & CISPR 22
Voltage	+5 V ~ +24 V
Voltage Regulation	±3% @ constant voltage mode
Maximum Power	70 W
Ripple & Noise	1%
Hold-up Time	10 ms typical at full load @ 115 VAC
Protection	Over Voltage Protection (OVP), AC recycle short circuit protection; Output short circuit (<0.03 Ω) Auto recover
Safety	UL 1950; CSA 22.2-234; TUV EN60950; I EC 950; CE EMC & LVD
Operating Temperature	0 to 40° C ambient
Storage Temperature	-10 to 70° C
MTBF	>100,000 hours at full load and 25° C ambient conditions
Cord Length	Shielded AC Input: 6′ (±2″) Output: 5′ (±1″)
Certification	CE, UL, CSA

Power Supply Output Connector Pinouts

Pin	Description
1	+24 V
2	RETURN

Picomotor

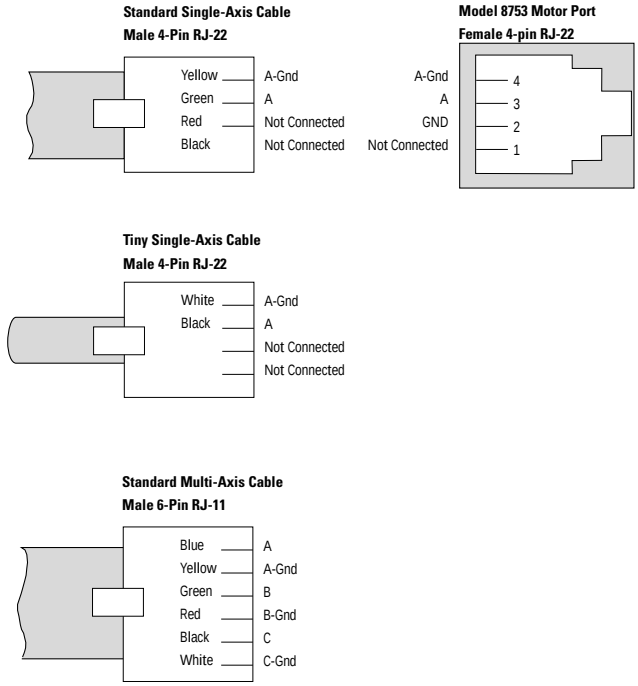
Picomotor Characteristics

Specifications	Standard MRA	Tiny MRA
Linear Resolution: (see Note)	<30 nm	<100 nm
Angular Resolution: (see Note)	<0.6 mrad	<2 mrad
Linear Travel:	Limited by screw length	Limited by screw length
Max. Load:	5 lbs (22 N)	1.5 lbs (6.7 N)
Max. Speed:	0.64 mm/min 2–3 RPM	1 mm/min 2–3 RPM
Temperature Range	10–40˚ C	10–40˚ C
Est. Lifetime| (continuous operation)	1000 hrs	200 hrs

Note: *Since the Picomotor relies on friction to turn the screw, the actual angle change and linear travel per pulse varies a small amount with direction of rotation, load, temperature, and life of the unit. See page 8 for more information on step size.*

4-Pin and 6-Pin Picomotor Pinouts

Figure 16: Wiring diagrams for 4- and 6-pin connectors



Vacuum-Compatible Connectors

Vacuum-compatible Picomotors are shipped without connectors attached. For these Picomotors, the red lead is the signal and the white lead is common.

Customer Service

Technical Support

Information and advice about the operation of any New Focus product is available from our applications engineers. For quickest response ask for "Technical Support" and know the model and serial number for your product.

Hours: 8:00–5:00 PST, Monday through Friday
(excluding holidays).

Toll Free: 1-866-NUFOCUS (1-866-683-6287)
(from the USA & Canada only)
Phone: (408) 284-6808

Support is also available by fax and email:

Fax: (408) 980-8883
Email: techsupport@newfocus.com

We typically respond to faxes and email within one business day.

Service

In the event that your product malfunctions or becomes damaged, please contact New Focus for a return authorization number and instructions on shipping the unit back for evaluation and repair.